

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

November 23, 2021

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 6.4 of `nicematrix`, at the date of 2021/11/23.

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}


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


```

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. 19

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 `hvlines` draws all the vertical and horizontal 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. 45;
- **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. 27).

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. 43):

```
\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

New 6.2

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 43.

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. 22).
- 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		

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. 44).

```

\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 & 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. 12).

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.

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.¹⁸

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.

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

¹⁸If you use Overleaf, Overleaf will do automatically the right number of compilations.

```

\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`.¹⁹

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.

```

\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. 20). 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

¹⁹Remark that, in the `\CodeBefore`, PGF/Tikz nodes of the form “(i-lj)” are also available to indicate the position to the potential rules: cf. p. 40.

We have used the key `r` which aligns all the columns rightwards (cf. p. 35).

- 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`²⁰. 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.²¹
- 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).

²⁰The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

²¹Otherwise, the color of a given row relies only upon the parity of its absolute number.

```

\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}{lrr}[hvlines]
\CodeBefore
  \rowcolors{1}{blue!10}{}[respect-blocks]
\Body
\Block{2-1}{John}      & 12 \\\
                        & 13 \\\
Steph                  & 8 \\\
\Block{3-1}{Sarah}     & 18 \\\
                        & 17 \\\
                        & 15 \\\
                        & 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

- 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` 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.²²

There are three commands available (they are inspired by `colortbl` but are *independent* of `colortbl`):

- `\cellcolor` which colorizes a cell;
- `\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

```

²²Up to now, this key is *not* available in `\NiceMatrixOptions`.


```

\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.²³
- **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. 35.

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`.

²³The 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).

```
\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. 20) 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.²⁴

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

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

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

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

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`²⁵. 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}
```

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

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

²⁴The 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).

²⁵At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

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}}
```

- | |
|---|
| <ul style="list-style-type: none"> • first item • second item |
|---|

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

- | |
|---|
| <ul style="list-style-type: none"> • 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. 38. 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}
```

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

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.²⁶ 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).²⁷
- 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[l]}[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”.

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

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

The dotted lines have been drawn with the tools presented p. 22.

²⁶If `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`).

²⁷The 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.

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.²⁸
- 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. 37) 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}}
$\begin{pNiceArray}{cc|cc}[first-row,last-row=5,first-col,last-col,nullify-dots]
    & C_1 & & \Cdots & & C_4 & & \\
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 & & \\
\vdots & a_{21} & a_{22} & a_{23} & a_{24} & \vdots & & \\
\hline
& a_{31} & a_{32} & a_{33} & a_{34} & & & \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 & & \\
& C_1 & & \Cdots & & C_4 & & \\
\end{pNiceArray}$
```

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

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 43.

²⁸The users wishing exteriors columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 28).

- 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. 17) 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. 28.

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`.²⁹

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells³⁰ 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`.³¹

```
\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}
```

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

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

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

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

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}
```

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

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

²⁹The 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`.

³⁰The precise definition of a “non-empty cell” is given below (cf. p. 44).

³¹It’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. 25.

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 & \\
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.³²

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}
```

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

10.1 The option `nullify-dots`

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

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

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

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.

³²In `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. 17

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`³³ 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`:

```
\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}
```

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

³³We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

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`.³⁴

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`²⁹ 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}
\end{pmatrix}
```

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

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. 27) 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}
```

$$\begin{bmatrix} 1 & & & 0 \\ \vdots & \ddots & & \\ 0 & & & 1 \end{bmatrix}$$

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. 27) 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:

³⁴The 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.

- `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. 20.

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).³⁵

```
\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 paths (with the exception of “`color`”, “`shorten >`” and “`shorten <`”).

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

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

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

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 `l` in the preamble, by the command `\Hline` and by the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` are not drawn within the blocks).³⁶

³⁵The 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.

³⁶On the other side, the command `\line` in the `\CodeAfter` (cf. p. 27) does *not* create block.

```

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

```

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

11 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.³⁷

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. 37.

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. 25).

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

```

\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}

```

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

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. 43).

```

\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}

```

$$\left[\begin{array}{cccccc|cccc} 1 & \cdots & & 1 & 2 & \cdots & & 2 & \\ 0 & \ddots & & \vdots & \vdots & \hspace*{2.5cm} & & \vdots & \\ \vdots & \ddots & & & & & & & \\ 0 & \cdots & 0 & 1 & 2 & \cdots & & 2 & \end{array} \right]$$

³⁷There is also a key `code-before` described p. 13.

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.³⁸

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.

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

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

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);

³⁸There 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}]`).

- **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.

```


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


```

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

```


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


```

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. 41.

It's also possible to specify some delimiters³⁹ 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.

```


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


```

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

³⁹Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

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}
```

$$\begin{array}{cccccc} & \overbrace{\hspace{1.5cm}}^A & & \overbrace{\hspace{1.5cm}}^B & & \\ \left(\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 11 & 12 & 13 & 14 & 15 & 16 \end{array} \right) \end{array}$$

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).

```
\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}
```

$$\begin{array}{cccccc} & \overbrace{\hspace{1.5cm}}^A & & \overbrace{\hspace{1.5cm}}^B & & \\ \left(\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 11 & 12 & 13 & 14 & 15 & 16 \end{array} \right) \end{array}$$

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. 32. 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}

```

Table 1: Use of `\tabularnote`^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.

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.

- 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`. The key `notes/enumitem-keys` specifies a list of pairs `key=value` (following the specifications of `enumitem`) to customize that type of list.

Initial value: `noitemsep , leftmargin = * , align = left , labelsep = Opt`

This initial value contains the specification `align = left` which 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. 32).

- 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`.

Initial value: `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. 46.

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`.

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

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

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).

```


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


```

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`.⁴⁰

```

\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}

```

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

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 & e_2 & 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`).

```


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


```

⁴⁰It can also be used in `\RowStyle` (cf. p. 17).

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

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 column⁴¹. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 13) and in the `\CodeAfter` (cf. p. 27), `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).

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

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

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).

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

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

⁴¹We 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.

13.6 The option `light-syntax`

The option `light-syntax` (inspired 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.

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

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

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.⁴²

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. 28).

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 an empty cell.

⁴²The 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.

However, it's possible to impose the creation of a node with the command `\NotEmpty`.⁴³

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.

```

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

```

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

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).

```

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

```

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

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 52).

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

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.

⁴³One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 22) and the computation of the “corners” (cf. p. 10).

```

\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. 40).

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`.⁴⁴

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

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.

$$\left(\begin{array}{ccc} a & a+b & a+b+c \\ \underline{a} & \underline{a} & \underline{a+b} \\ a & a & a \end{array} \right)$$

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.⁴⁵

$$\left(\begin{array}{ccc} a & a+b & a+b+c \\ a & a & a+b \\ \underline{a} & \underline{a} & \underline{a} \end{array} \right)$$

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`.⁴⁶

$$\left(\begin{array}{ccc} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{array} \right)$$

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

⁴⁴There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

⁴⁵There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 20).

⁴⁶The 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.

fix both values with the option `extra-margin` and, in the following example, we use `extra-margin` with the value 3 pt.

$$\left(\begin{array}{|c|c|c|} \hline a & a+b & a+b+c \\ \hline a & a & a+b \\ \hline a & a & a \\ \hline \end{array} \right)$$

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 & brugnon
\end{NiceTabular}
```

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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. 13). 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 contruction of the array itself).

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).

```

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

```

$$\begin{pmatrix} 2 & 1 & 3 & | & 0 \\ 3 & 3 & 1 & | & 0 \\ \hline 3 & 3 & 1 & | & 0 \end{pmatrix}$$

14.4 The nodes corresponding to the command `\SubMatrix`

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 28.

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}\}`).

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

15 API for the developpers

The package `nicematrix` provides two variables which are internal but public⁴⁷:

- `\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.

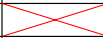
It’s possible to program such command `\crossbox` as follows, explicitly using the public variable `\g_nicematrix_code_after_tl`.

```
\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

⁴⁷According 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 job⁴⁸:

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

The heavy vertical rule won't extend in the exterior rows.⁴⁹

```
\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 lines⁵⁰ 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:

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

⁴⁸The command `\vrule` is a TeX (and not LaTeX) command.

⁴⁹Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

⁵⁰We 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. 19) 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 idea⁵¹. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`⁵². 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).

⁵¹In 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).*

⁵²And 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).⁵³

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.⁵⁴
For the following example, you 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>	

⁵³See <https://tex.stackexchange.com/questions/528975/error-loading-tikz-in-ieeeaccess-class>

⁵⁴By 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. 30.

Let's consider that we wish to number the notes of a tabular with stars.⁵⁵

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 ⁵⁶

```
\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}
```

⁵⁵Of course, it's realistic only when there is very few notes in the tabular.

⁵⁶In 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 & 0 \\
\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}}
\[ \begin{pNiceMatrix}[last-row,last-col,nullify-dots,xdots/line-style={dashed,blue}]
1& & \Vdots & & & \Vdots \\
& \Ddots[line-style=standard] & & & & \\
& & 1 & & & \\
& \Cdots[color=blue,line-style=dashed]& & & \blue 0 & \\
& \Cdots & & & \blue 1 & & \Cdots & \blue \leftarrow i \\
& & & & 1 & & \\
& & & \Vdots & & \Ddots[line-style=standard] & & \Vdots \\
& & & & & & 1 & \\
& \Cdots & & & \blue 1 & \Cdots & & \Cdots & \blue 0 & & \Cdots & \blue \leftarrow j \\
& & & & & & & & 1 & & \\
& & & & & & & \Ddots[line-style=standard] & & & \\
& & & \Vdots & & & & \Vdots & & & 1 & \\
& & & \blue \overset{\uparrow}{i} & & & & \blue \overset{\uparrow}{j} & & & \\
\end{pNiceMatrix} \]

```

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

In fact, it's even possible to draw solid lines with the commands `\Cdots`, `\Vdots`, etc.⁵⁷

```

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

```

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

⁵⁷In 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.

New 6.2 In fact, it's possible, with the key `vlines-in-sub-matrix`, to choice 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}\]

```

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

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 block⁵⁸).

```

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

```

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

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.⁵⁹

⁵⁸We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

⁵⁹For 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 \AtBeginDocument { \bool_set_false:N \c_@@_in_preamble_bool }

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

```

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.


```

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

```

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).

```

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

```

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

```

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

66 \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.

```

67 \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.

```

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

```

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.

```

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

```

```
92 }
```

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.

```
93 \AtBeginDocument
94 {
95   \ifpackageloaded { booktabs }
96     { \iow_now:Nn \mainaux \nicematrix@redefine@check@rerun }
97     { }
98   }
99 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
100 {
101   \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`).

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

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

```
108 \bool_new:N \c_@@_colortbl_loaded_bool
109 \AtBeginDocument
110 {
111   \ifpackageloaded { colortbl }
112     { \bool_set_true:N \c_@@_colortbl_loaded_bool }
113     { }
```

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.

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

Idem for `\CT@drs@`.

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

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.

```

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

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`⁶⁰

```

148   \skip_horizontal:N \c_zero_dim
149 }

```

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`.

```

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

```

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.

```

154 \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`.

```

155 { \@@_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*.

```

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

```

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).

```

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

```

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

```

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

```

⁶⁰See question 99041 on TeX StackExchange.

```

178 }
179 \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.

```

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

```

The following command is a small shortcut.

```

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

```

```

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

```

```

190 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

```

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.

```

191 \bool_new:N \c_@@_siunitx_loaded_bool
192 \AtBeginDocument
193 {
194   \ifpackageloaded { siunitx }
195   { \bool_set_true:N \c_@@_siunitx_loaded_bool }
196   { }
197 }

```

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

```

198 \AtBeginDocument
199 {
200   \bool_if:nTF { ! \c_@@_siunitx_loaded_bool }
201   { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
202   {
203     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
204     {
205       \renewcommand*{\NC@rewrite@S}[1] []
206       {

```

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

```

207       \@temptokena \exp_after:wN
208       { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
209       \NC@find
210     }
211   }
212 }
213 }

```

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.

```

214 \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).

```
215 \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.

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

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

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

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

```
220 \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).

```
221 \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.).

```
222 \dim_new:N \l_@@_col_width_dim
223 \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.

```
224 \int_new:N \g_@@_row_total_int
225 \int_new:N \g_@@_col_total_int
```

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

```
226 \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.

```
227 \str_new:N \l_@@_hpos_cell_str
228 \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`.

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

Idem pour the mono-row blocks.

```
230 \dim_new:N \g_@@_blocks_ht_dim
231 \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}`).

```
232 \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`.

```
233 \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.

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

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

```
235 \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.

```
236 \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.

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

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

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

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

```
239 \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.

```
240 \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 contain of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }`).

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

```
242 \cs_new_protected:Npn \@@_test_if_math_mode:
243 {
244   \if_mode_math: \else:
245     \@@_fatal:n { Outside~math~mode }
246   \fi:
247 }
```

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

```
248 \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.

```
249 \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”.

```
250 \colorlet { nicematrix-last-col } { . }
251 \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`).

```
252 \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*.

```
253 \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).

```
254 \cs_new:Npn \@@_full_name_env:
255 {
256   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
257   { command \space \c_backslash_str \g_@@_name_env_str }
258   { environment \space \{ \g_@@_name_env_str \} }
259 }
```

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*.

```
260 \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.

```
261 \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.

```
262 \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.

```
263 \int_new:N \l_@@_old_iRow_int
264 \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.

```
265 \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.

```
266 \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.

```
267 \bool_new:N \l_@@_X_columns_aux_bool
268 \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.

```
269 \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).

```
270 \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).

```
271 \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`).

```
272 \tl_new:N \l_@@_code_before_tl
273 \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).

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

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

```
275 \dim_new:N \l_@@_x_initial_dim
276 \dim_new:N \l_@@_y_initial_dim
277 \dim_new:N \l_@@_x_final_dim
278 \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 (if they don't exist yet: that's why we use `\dim_zero_new:N`).

```
279 \dim_zero_new:N \l_tmpc_dim
280 \dim_zero_new:N \l_tmpd_dim
```

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

```
281 \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”.

```
282 \dim_new:N \g_@@_width_last_col_dim
283 \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.

```
284 \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!).

```
285 \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}`.

```
286 \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.

```
287 \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.

```
288 \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`).

```
289 \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.

```
290 \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).

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

```
292 \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`).

```
293 \int_new:N \l_@@_row_min_int
```

```
294 \int_new:N \l_@@_row_max_int
```

```
295 \int_new:N \l_@@_col_min_int
```

```
296 \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.

```
297 \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).

```
298 \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`.

```
299 \tl_new:N \l_@@_fill_tl
300 \tl_new:N \l_@@_draw_tl
301 \seq_new:N \l_@@_tikz_seq
302 \clist_new:N \l_@@_borders_clist
303 \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`.

```
304 \tl_new:N \l_@@_color_tl
```

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

```
305 \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`).

```
306 \str_new:N \l_@@_hpos_block_str
307 \str_set:Nn \l_@@_hpos_block_str { c }
308 \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`.

```
309 \tl_new:N \l_@@_vpos_of_block_tl
310 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

```
311 \bool_new:N \l_@@_draw_first_bool
```

The following flag corresponds to the key `hvlines` of the command `\Block`.

```
312 \bool_new:N \l_@@_hvlines_block_bool
```

The blocks which use the key `-` will store their content in a box. These boxes are numbered with the following counter.

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

314 \dim_new:N \l_@@_submatrix_extra_height_dim
315 \dim_new:N \l_@@_submatrix_left_xshift_dim
316 \dim_new:N \l_@@_submatrix_right_xshift_dim
317 \clist_new:N \l_@@_hlines_clist
318 \clist_new:N \l_@@_vlines_clist
319 \clist_new:N \l_@@_submatrix_hlines_clist
320 \clist_new:N \l_@@_submatrix_vlines_clist
```

The following flag will be used by (for instance) `\l_@@_vline_ii:nmmn`. When `\l_@@_dotted_bool` is `true`, a dotted line (with our system) will be drawn.

```
321 \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.

```
322 \int_new:N \l_@@_first_row_int
323 \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.

```
324 \int_new:N \l_@@_first_col_int
325 \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).

```
326 \int_new:N \l_@@_last_row_int
327 \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”.⁶¹

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

Idem for `\l_@@_last_col_without_value_bool`

```
329 \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.

```
330 \int_new:N \l_@@_last_col_int
331 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

⁶¹We 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.

```

\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.

```

332 \bool_new:N \g_@@_last_col_found_bool

```

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

Some utilities

```

333 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
334 {
335   \tl_set:Nn \l_tmpa_tl { #1 }
336   \tl_set:Nn \l_tmpb_tl { #2 }
337 }

```

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.

```

338 \cs_new_protected:Npn \@@_expand_clist:N #1
339 {
340   \clist_if_in:NnF #1 { all }
341   {
342     \clist_clear:N \l_tmpa_clist
343     \clist_map_inline:Nn #1
344     {
345       \tl_if_in:nnTF { ##1 } { - }
346       { \@@_cut_on_hyphen:w ##1 \q_stop }
347       {
348         \tl_set:Nn \l_tmpa_tl { ##1 }
349         \tl_set:Nn \l_tmpb_tl { ##1 }
350       }
351       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
352       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
353     }
354     \tl_set_eq:NN #1 \l_tmpa_clist
355   }
356 }

```

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.

```

357 \newcounter { tablarnote }

```

We will store in the following sequence the tabular notes of a given array.

```

358 \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.

```

359 \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).

```
360 \int_new:N \l_@@_number_of_notes_int
```

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

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

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

```
362 \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`.

```
363 \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`.

```
364 \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`).

```
365 \AtBeginDocument
366 {
367   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
368   {
369     \NewDocumentCommand \tabularnote { m }
370     { \@@_error:n { enumitem-not-loaded } }
371   }
372 }
```

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.

```
373 \newlist { tabularnotes } { enumerate } { 1 }
374 \setlist [ tabularnotes ]
375 {
376   topsep = 0pt ,
377   noitemsep ,
378   leftmargin = * ,
379   align = left ,
380   labelsep = 0pt ,
381   label =
382     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
383 }
384 \newlist { tabularnotes* } { enumerate* } { 1 }
385 \setlist [ tabularnotes* ]
386 {
387   afterlabel = \nobreak ,
388   itemjoin = \quad ,
389   label =
390     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
391 }
```

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.⁶²

```

392     \NewDocumentCommand \tabularnote { m }
393     {
394         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
395             { \@@_error:n { tabularnote~forbidden } }
396             {

```

`\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).

```

397         \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`.

```

398         \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
399         \peek_meaning:NF \tabularnote
400         {

```

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.

```

401         \hbox_set:Nn \l_tmpa_box
402         {

```

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

```

403         \@@_notes_label_in_tabular:n
404         {
405             \stepcounter { tabularnote }
406             \@@_notes_style:n { tabularnote }
407             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
408             {
409                 ,
410                 \stepcounter { tabularnote }
411                 \@@_notes_style:n { tabularnote }
412             }
413         }
414     }

```

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.

```

415         \addtocounter { tabularnote } { -1 }
416         \refstepcounter { tabularnote }
417         \int_zero:N \l_@@_number_of_notes_int
418         \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).

```

419         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
420     }
421 }
422 }
423 }
424 }

```

⁶²We 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.

```

425 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
426 {
427   \begin { pgfscope }
428   \pgfset
429   {
430     outer~sep = \c_zero_dim ,
431     inner~sep = \c_zero_dim ,
432     minimum~size = \c_zero_dim
433   }
434   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
435   \pgfnode
436   { rectangle }
437   { center }
438   {
439     \vbox_to_ht:nn
440     { \dim_abs:n { #5 - #3 } }
441     {
442       \vfill
443       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
444     }
445   }
446   { #1 }
447   { }
448   \end { pgfscope }
449 }
```

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.

```

450 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
451 {
452   \begin { pgfscope }
453   \pgfset
454   {
455     outer~sep = \c_zero_dim ,
456     inner~sep = \c_zero_dim ,
457     minimum~size = \c_zero_dim
458   }
459   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
460   \pgfpointdiff { #3 } { #2 }
461   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
462   \pgfnode
463   { rectangle }
464   { center }
465   {
466     \vbox_to_ht:nn
467     { \dim_abs:n \l_tmpb_dim }
468     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
469   }
470   { #1 }
471   { }
472   \end { pgfscope }
473 }
```

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.

```
474 \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`.

```
475 \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`).

```
476 \dim_new:N \l_@@_cell_space_top_limit_dim
477 \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.

```
478 \dim_new:N \l_@@_inter_dots_dim
479 \AtBeginDocument { \dim_set:Nn \l_@@_inter_dots_dim { 0.45 em } }
```

The `\AtBeginDocument` is only a 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).

```
480 \dim_new:N \l_@@_xdots_shorten_dim
481 \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 em } }
```

The `\AtBeginDocument` is only a 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.

```
482 \dim_new:N \l_@@_radius_dim
483 \AtBeginDocument { \dim_set:Nn \l_@@_radius_dim { 0.53 pt } }
```

The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even if 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.

```
484 \tl_new:N \l_@@_xdots_line_style_tl
485 \tl_const:Nn \c_@@_standard_tl { standard }
486 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax`.

```
487 \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).

```
488 \tl_new:N \l_@@_baseline_tl
489 \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`).

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

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

```
491 \bool_new:N \l_@@_parallelize_diags_bool
492 \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.

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

```
494 \dim_new:N \l_@@_notes_above_space_dim
495 \AtBeginDocument { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even if 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.).

```
496 \bool_new:N \l_@@_nullify_dots_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).

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

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

```
498 \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.

```
499 \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”.

```
500 \bool_new:N \l_@@_medium_nodes_bool
501 \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).

```
502 \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`).

```
503 \dim_new:N \l_@@_left_margin_dim
504 \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`.

```
505 \dim_new:N \l_@@_extra_left_margin_dim
506 \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.

```
507 \tl_new:N \l_@@_end_of_row_tl
508 \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 “:”.

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

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

```
510 \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 fonction of its size. That’s why we create an option called `delimiters/max-width` which will give to the delimiters the width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

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

```
512 \keys_define:nn { NiceMatrix / xdots }
513 {
514   line-style .code:n =
515   {
516     \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.

```
517   { \cs_if_exist_p:N \tikzpicture }
518   { \str_if_eq_p:nn { #1 } { standard } }
519   { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
520   { \@@_error:n { bad-option-for~line-style } }
521   } ,
522   line-style .value_required:n = true ,
523   color .tl_set:N = \l_@@_xdots_color_tl ,
524   color .value_required:n = true ,
525   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
526   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 `_`.

```
527   down .tl_set:N = \l_@@_xdots_down_tl ,
528   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).

```
529   draw-first .code:n = \prg_do_nothing: ,
530   unknown .code:n = \@@_error:n { Unknown-key-for~xdots }
531 }
```

```
532 \keys_define:nn { NiceMatrix / rules }
533 {
534   color .tl_set:N = \l_@@_rules_color_tl ,
535   color .value_required:n = true ,
536   width .dim_set:N = \arrayrulewidth ,
537   width .value_required:n = true
538 }
```

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

```

539 \keys_define:nn { NiceMatrix / Global }
540 {
541   delimiters .code:n =
542     \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
543   delimiters .value_required:n = true ,
544   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
545   rules .value_required:n = true ,
546   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
547   standard-cline .default:n = true ,
548   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
549   cell-space-top-limit .value_required:n = true ,
550   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
551   cell-space-bottom-limit .value_required:n = true ,
552   cell-space-limits .meta:n =
553     {
554       cell-space-top-limit = #1 ,
555       cell-space-bottom-limit = #1 ,
556     } ,
557   cell-space-limits .value_required:n = true ,
558   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
559   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
560   light-syntax .default:n = true ,
561   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
562   end-of-row .value_required:n = true ,
563   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
564   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
565   last-row .int_set:N = \l_@@_last_row_int ,
566   last-row .default:n = -1 ,
567   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
568   code-for-first-col .value_required:n = true ,
569   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
570   code-for-last-col .value_required:n = true ,
571   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
572   code-for-first-row .value_required:n = true ,
573   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
574   code-for-last-row .value_required:n = true ,
575   hlines .clist_set:N = \l_@@_hlines_clist ,
576   vlines .clist_set:N = \l_@@_vlines_clist ,
577   hlines .default:n = all ,
578   vlines .default:n = all ,
579   vlines-in-sub-matrix .code:n =
580     {
581       \tl_if_single_token:nTF { #1 }
582         { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
583         { \@@_error:n { One-letter~allowed } }
584     } ,
585   vlines-in-sub-matrix .value_required:n = true ,
586   hvlines .code:n =
587     {
588       \clist_set:Nn \l_@@_vlines_clist { all }
589       \clist_set:Nn \l_@@_hlines_clist { all }
590     } ,
591   hvlines-except-borders .code:n =
592     {
593       \clist_set:Nn \l_@@_vlines_clist { all }
594       \clist_set:Nn \l_@@_hlines_clist { all }
595       \bool_set_true:N \l_@@_except_borders_bool
596     } ,
597   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.

```

598   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
599   renew-dots .value_forbidden:n = true ,
600   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
601   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
602   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
603   create-extra-nodes .meta:n =
604     { create-medium-nodes , create-large-nodes } ,
605   left-margin .dim_set:N = \l_@@_left_margin_dim ,
606   left-margin .default:n = \arraycolsep ,
607   right-margin .dim_set:N = \l_@@_right_margin_dim ,
608   right-margin .default:n = \arraycolsep ,
609   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
610   margin .default:n = \arraycolsep ,
611   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
612   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
613   extra-margin .meta:n =
614     { extra-left-margin = #1 , extra-right-margin = #1 } ,
615   extra-margin .value_required:n = true ,
616 }

```

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

```

617 \keys_define:nn { NiceMatrix / Env }
618 {

```

The key `hvlines-except-corners` is now deprecated (use `hvlines` and `corners` instead).

```

619   hvlines-except-corners .code:n =
620     {
621       \clist_set:Nn \l_@@_corners_clist { #1 }
622       \clist_set:Nn \l_@@_vlines_clist { all }
623       \clist_set:Nn \l_@@_hlines_clist { all }
624     } ,
625   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
626   corners .clist_set:N = \l_@@_corners_clist ,
627   corners .default:n = { NW , SW , NE , SE } ,
628   code-before .code:n =
629     {
630       \tl_if_empty:nF { #1 }
631       {
632         \tl_put_right:Nn \l_@@_code_before_tl { #1 }
633         \bool_set_true:N \l_@@_code_before_bool
634       }
635     } ,

```

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

```

636   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
637   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
638   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
639   baseline .tl_set:N = \l_@@_baseline_tl ,
640   baseline .value_required:n = true ,
641   columns-width .code:n =
642     \tl_if_eq:nnTF { #1 } { auto }
643     { \bool_set_true:N \l_@@_auto_columns_width_bool }
644     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
645   columns-width .value_required:n = true ,
646   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.

```

647   \legacy_if:nF { measuring@ }

```

```

648     {
649         \str_set:Nn \l_tmpa_str { #1 }
650         \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
651         { \@@_error:nn { Duplicate~name } { #1 } }
652         { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
653         \str_set_eq:NN \l_@@_name_str \l_tmpa_str
654     } ,
655     name .value_required:n = true ,
656     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
657     code-after .value_required:n = true ,
658     colortbl-like .code:n =
659         \bool_set_true:N \l_@@_colortbl_like_bool
660         \bool_set_true:N \l_@@_code_before_bool ,
661     colortbl-like .value_forbidden:n = true
662 }
663 \keys_define:nn { NiceMatrix / notes }
664 {
665     para .bool_set:N = \l_@@_notes_para_bool ,
666     para .default:n = true ,
667     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
668     code-before .value_required:n = true ,
669     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
670     code-after .value_required:n = true ,
671     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
672     bottomrule .default:n = true ,
673     style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
674     style .value_required:n = true ,
675     label-in-tabular .code:n =
676         \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
677     label-in-tabular .value_required:n = true ,
678     label-in-list .code:n =
679         \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
680     label-in-list .value_required:n = true ,
681     enumitem-keys .code:n =
682     {
683         \bool_if:NTF \c_@@_in_preamble_bool
684         {
685             \AtBeginDocument
686             {
687                 \bool_if:NT \c_@@_enumitem_loaded_bool
688                 { \setlist* [ tabularnotes ] { #1 } }
689             }
690         }
691         {
692             \bool_if:NT \c_@@_enumitem_loaded_bool
693             { \setlist* [ tabularnotes ] { #1 } }
694         }
695     } ,
696     enumitem-keys .value_required:n = true ,
697     enumitem-keys-para .code:n =
698     {
699         \bool_if:NTF \c_@@_in_preamble_bool
700         {
701             \AtBeginDocument
702             {
703                 \bool_if:NT \c_@@_enumitem_loaded_bool
704                 { \setlist* [ tabularnotes* ] { #1 } }
705             }
706         }
707         {
708             \bool_if:NT \c_@@_enumitem_loaded_bool
709             { \setlist* [ tabularnotes* ] { #1 } }
710         }
711     }

```

```

711     } ,
712     enumitem-keys-para .value_required:n = true ,
713     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
714 }
715 \keys_define:nn { NiceMatrix / delimiters }
716 {
717     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
718     max-width .default:n = true ,
719     color .tl_set:N = \l_@@_delimiters_color_tl ,
720     color .value_required:n = true ,
721 }

```

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

```

722 \keys_define:nn { NiceMatrix }
723 {
724     NiceMatrixOptions .inherit:n =
725     { NiceMatrix / Global } ,
726     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
727     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
728     NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
729     NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
730     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
731     SubMatrix / rules .inherit:n = NiceMatrix / rules ,
732     CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
733     NiceMatrix .inherit:n =
734     {
735         NiceMatrix / Global ,
736         NiceMatrix / Env ,
737     } ,
738     NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
739     NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
740     NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
741     NiceTabular .inherit:n =
742     {
743         NiceMatrix / Global ,
744         NiceMatrix / Env
745     } ,
746     NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
747     NiceTabular / rules .inherit:n = NiceMatrix / rules ,
748     NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
749     NiceArray .inherit:n =
750     {
751         NiceMatrix / Global ,
752         NiceMatrix / Env ,
753     } ,
754     NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
755     NiceArray / rules .inherit:n = NiceMatrix / rules ,
756     NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
757     pNiceArray .inherit:n =
758     {
759         NiceMatrix / Global ,
760         NiceMatrix / Env ,
761     } ,
762     pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
763     pNiceArray / rules .inherit:n = NiceMatrix / rules ,
764     pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
765 }

```

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

```

766 \keys_define:nn { NiceMatrix / NiceMatrixOptions }

```

```

767 {
768   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
769   width .value_required:n = true ,
770   last-col .code:n = \tl_if_empty:nF { #1 }
771     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
772     \int_zero:N \l_@@_last_col_int ,
773   small .bool_set:N = \l_@@_small_bool ,
774   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.

```

775   renew-matrix .code:n = \@@_renew_matrix: ,
776   renew-matrix .value_forbidden:n = true ,

```

The key `transparent` is now considered as obsolete (because its name is ambiguous).

```

777   transparent .code:n =
778     {
779       \@@_renew_matrix:
780       \bool_set_true:N \l_@@_renew_dots_bool
781       \@@_error:n { Key~transparent }
782     } ,
783   transparent .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}`.

```

784   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.

```

785   columns-width .code:n =
786     \tl_if_eq:nnTF { #1 } { auto }
787     { \@@_error:n { Option~auto~for~columns-width } }
788     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

Usually, an error is raised when the user tries to give the same name to two distinct environments of `nicematrix` (theses names are global and not local to the current TeX scope). However, the option `allow-duplicate-names` disables this feature.

```

789   allow-duplicate-names .code:n =
790     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
791   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`).

```

792   letter-for-dotted-lines .code:n =
793     {
794       \tl_if_single_token:nTF { #1 }
795       { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
796       { \@@_error:n { One-letter~allowed } }
797     } ,
798   letter-for-dotted-lines .value_required:n = true ,
799   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
800   notes .value_required:n = true ,
801   sub-matrix .code:n =
802     \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
803   sub-matrix .value_required:n = true ,
804   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
805 }

806 \str_new:N \l_@@_letter_for_dotted_lines_str
807 \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.

```
808 \NewDocumentCommand \NiceMatrixOptions { m }
809 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }
```

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

```
810 \keys_define:nn { NiceMatrix / NiceMatrix }
811 {
812   last-col .code:n = \tl_if_empty:nTF {#1}
813     {
814       \bool_set_true:N \l_@@_last_col_without_value_bool
815       \int_set:Nn \l_@@_last_col_int { -1 }
816     }
817     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
818   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
819   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
820   small .bool_set:N = \l_@@_small_bool ,
821   small .value_forbidden:n = true ,
822   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
823 }
```

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

```
824 \keys_define:nn { NiceMatrix / NiceArray }
825 {
```

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.

```
826   small .bool_set:N = \l_@@_small_bool ,
827   small .value_forbidden:n = true ,
828   last-col .code:n = \tl_if_empty:nF { #1 }
829     { \@@_error:n { last-col-non-empty-for-NiceArray } }
830     { \int_zero:N \l_@@_last_col_int ,
831   notes / para .bool_set:N = \l_@@_notes_para_bool ,
832   notes / para .default:n = true ,
833   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
834   notes / bottomrule .default:n = true ,
835   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
836   tabularnote .value_required:n = true ,
837   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
838   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
839   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
840 }
```

```
841 \keys_define:nn { NiceMatrix / pNiceArray }
842 {
843   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
844   last-col .code:n = \tl_if_empty:nF {#1}
845     { \@@_error:n { last-col-non-empty-for-NiceArray } }
846     { \int_zero:N \l_@@_last_col_int ,
847   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
848   small .bool_set:N = \l_@@_small_bool ,
849   small .value_forbidden:n = true ,
850   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
851   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
852   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
853 }
```

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

```
854 \keys_define:nn { NiceMatrix / NiceTabular }
855 {
```


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.

```

856   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
857           \bool_set_true:N \l_@@_width_used_bool ,
858   width .value_required:n = true ,
859   notes / para .bool_set:N = \l_@@_notes_para_bool ,
860   notes / para .default:n = true ,
861   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
862   notes / bottomrule .default:n = true ,
863   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
864   tabularnote .value_required:n = true ,
865   last-col .code:n = \tl_if_empty:NF {#1}
866           { \@@_error:n { last-col~non-empty-for~NiceArray } }
867           \int_zero:N \l_@@_last_col_int ,
868   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
869   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
870   unknown .code:n = \@@_error:n { Unknown-key-for~NiceTabular }
871 }

```

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}`).

```

872 \cs_new_protected:Npn \@@_cell_begin:w
873 {

```

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*).

```

874   \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).

```

875   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

We increment `\c@jCol`, which is the counter of the columns.

```

876   \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.

```

877   \int_compare:nNnT \c@jCol = 1
878     { \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).

```

879   \hbox_set:Nw \l_@@_cell_box
880   \bool_if:NF \l_@@_NiceTabular_bool
881   {
882     \c_math_toggle_token
883     \bool_if:NT \l_@@_small_bool \scriptstyle
884   }

```

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).

```

885   \color { nicematrix }
886   \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.

```

887 \int_compare:nNnTF \c@iRow = 0
888 {
889   \int_compare:nNnT \c@jCol > 0
890   {
891     \l_@@_code_for_first_row_tl
892     \xglobal \colorlet { nicematrix-first-row } { . }
893   }
894 }
895 {
896   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
897   {
898     \l_@@_code_for_last_row_tl
899     \xglobal \colorlet { nicematrix-last-row } { . }
900   }
901 }
902 }
```

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.

```

903 \cs_new_protected:Npn \@@_begin_of_row:
904 {
905   \int_gincr:N \c@iRow
906   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
907   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
908   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
909   \pgfpicture
910   \pgfrememberpicturepositiononpagetrue
911   \pgfcoordinate
912   { \@@_env: - row - \int_use:N \c@iRow - base }
913   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
914   \str_if_empty:NF \l_@@_name_str
915   {
916     \pgfnodealias
917     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
918     { \@@_env: - row - \int_use:N \c@iRow - base }
919   }
920   \endpgfpicture
921 }
```

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.

```

922 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
923 {
924   \int_compare:nNnTF \c@iRow = 0
925   {
926     \dim_gset:Nn \g_@@_dp_row_zero_dim
927     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
928     \dim_gset:Nn \g_@@_ht_row_zero_dim
929     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
930   }
931   {
932     \int_compare:nNnT \c@iRow = 1
933     {
934       \dim_gset:Nn \g_@@_ht_row_one_dim
```

```

935         { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
936     }
937 }
938 }
939 \cs_new_protected:Npn \@@_rotate_cell_box:
940 {
941     \box_rotate:Nn \l_@@_cell_box { 90 }
942     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
943     {
944         \vbox_set_top:Nn \l_@@_cell_box
945         {
946             \vbox_to_zero:n { }
947             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
948             \box_use:N \l_@@_cell_box
949         }
950     }
951     \bool_gset_false:N \g_@@_rotate_bool
952 }
953 \cs_new_protected:Npn \@@_adjust_size_box:
954 {
955     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
956     {
957         \box_set_wd:Nn \l_@@_cell_box
958         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
959         \dim_gzero:N \g_@@_blocks_wd_dim
960     }
961     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
962     {
963         \box_set_dp:Nn \l_@@_cell_box
964         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
965         \dim_gzero:N \g_@@_blocks_dp_dim
966     }
967     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
968     {
969         \box_set_ht:Nn \l_@@_cell_box
970         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
971         \dim_gzero:N \g_@@_blocks_ht_dim
972     }
973 }
974 \cs_new_protected:Npn \@@_cell_end:
975 {
976     \@@_math_toggle_token:
977     \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.

```

978     \g_@@_post_action_cell_tl
979     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
980     \@@_adjust_size_box:
981     \box_set_ht:Nn \l_@@_cell_box
982     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
983     \box_set_dp:Nn \l_@@_cell_box
984     { \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”).

```

985     \dim_gset:Nn \g_@@_max_cell_width_dim
986     { \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”.

```

987     \@@_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.

```

988 \bool_if:NTF \g_@@_empty_cell_bool
989   { \box_use_drop:N \l_@@_cell_box }
990   {
991     \bool_lazy_or:nnTF
992       \g_@@_not_empty_cell_bool
993       { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
994       \@@_node_for_cell:
995       { \box_use_drop:N \l_@@_cell_box }
996   }
997 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
998 \bool_gset_false:N \g_@@_empty_cell_bool
999 \bool_gset_false:N \g_@@_not_empty_cell_bool
1000 }
```

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

```

1001 \cs_new_protected:Npn \@@_node_for_cell:
1002 {
1003   \pgfpicture
1004   \pgfsetbaseline \c_zero_dim
1005   \pgfrememberpicturepositiononpagetrue
1006   \pgfset
1007   {
1008     inner~sep = \c_zero_dim ,
1009     minimum~width = \c_zero_dim
1010   }
1011   \pgfnode
1012   { rectangle }
1013   { base }
1014   { \box_use_drop:N \l_@@_cell_box }
1015   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1016   { }
1017   \str_if_empty:NF \l_@@_name_str
1018   {
1019     \pgfnodealias
1020     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1021     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1022   }
1023   \endpgfpicture
1024 }
```

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.

```

1025 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
```

```

1026 {
1027   \cs_new_protected:Npn \@@_patch_node_for_cell:
1028   {
1029     \hbox_set:Nn \l_@@_cell_box
1030     {
1031       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1032       \hbox_overlap_left:n
1033       {
1034         \pgfsys@markposition
1035         { \@@_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.

```

1036       #1
1037     }
1038     \box_use:N \l_@@_cell_box
1039     \box_move_down:nn { \box_dp: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 - SE }
1044       #1
1045     }
1046   }
1047 }
1048 }

```

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

```

1049 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1050 {
1051   \@@_patch_node_for_cell:n
1052   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1053 }
1054 { \@@_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.

```

1055 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1056 {
1057   \bool_if:nTF { #1 } { \tl_gput_left:cx \tl_gput_right:cx
1058     { \g_@@_#2 _ lines _ tl }
1059     {
1060       \use:c { @@ _ draw _ #2 : nnn }
1061       { \int_use:N \c@iRow }
1062       { \int_use:N \c@jCol }
1063       { \exp_not:n { #3 } }

```

```

1064     }
1065   }
1066   \cs_new_protected:Npn \@@_array:
1067   {
1068     \bool_if:NTF \l_@@_NiceTabular_bool
1069     { \dim_set_eq:NN \col@sep \tabcolsep }
1070     { \dim_set_eq:NN \col@sep \arraycolsep }
1071     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1072     { \cs_set_nopar:Npn \@halignto { } }
1073     { \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.

```

1074   \@tabarray
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose the
\array (of array) with the option t and the right translation will be done further. Remark that
\str_if_eq:VnTF is fully expandable and you need something fully expandable here.
1075   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1076   }

```

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.

```

1077 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1078 \cs_new_protected:Npn \@@_create_row_node:
1079   {

```

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

```

1080     \hbox
1081     {
1082       \bool_if:NT \l_@@_code_before_bool
1083       {
1084         \vtop
1085         {
1086           \skip_vertical:N 0.5\arrayrulewidth
1087           \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
1088           \skip_vertical:N -0.5\arrayrulewidth
1089         }
1090       }
1091       \pgfpicture
1092       \pgfrememberpicturerepositiononpagetrue
1093       \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
1094       { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1095       \str_if_empty:NF \l_@@_name_str
1096       {
1097         \pgfnodealias
1098         { \l_@@_name_str - row - \@@_succ:n \c@iRow }
1099         { \@@_env: - row - \@@_succ:n \c@iRow }
1100       }
1101       \endpgfpicture
1102     }
1103   }

```

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

```

1104 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1105 \cs_new_protected:Npn \@@_everycr_i:
1106   {
1107     \int_gzero:N \c@jCol
1108     \bool_gset_false:N \g_@@_after_col_zero_bool
1109     \bool_if:NF \g_@@_row_of_col_done_bool
1110     {
1111       \@@_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).

```

1112 \tl_if_empty:NF \l_@@_hlines_clist
1113 {
1114   \tl_if_eq:NnF \l_@@_hlines_clist { all }
1115   {
1116     \exp_args:NNx
1117     \clist_if_in:NnT
1118     \l_@@_hlines_clist
1119     { \@@_succ:n \c@iRow }
1120   }
1121 {

```

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.

```

1122 \int_compare:nNnT \c@iRow > { -1 }
1123 {
1124   \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@`.

```

1125   { \hrule height \arrayrulewidth width \c_zero_dim }
1126   }
1127 }
1128 }
1129 }
1130 }

```

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`).

```

1131 \cs_set_protected:Npn \@@_newcolumntype #1
1132 {
1133   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1134   \peek_meaning:NTF [
1135     { \newcol@ #1 }
1136     { \newcol@ #1 [ 0 ] }
1137   }

```

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

```

1138 \cs_set_protected:Npn \@@_renew_dots:
1139 {
1140   \cs_set_eq:NN \ldots \@@_Ldots
1141   \cs_set_eq:NN \cdots \@@_Cdots
1142   \cs_set_eq:NN \vdots \@@_Vdots
1143   \cs_set_eq:NN \ddots \@@_Ddots
1144   \cs_set_eq:NN \iddots \@@_Iddots
1145   \cs_set_eq:NN \dots \@@_Ldots
1146   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1147 }

```

When the key `colortbl-like` is used, the following code will be executed.

```

1148 \cs_new_protected:Npn \@@_colortbl_like:
1149 {
1150   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1151   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1152   \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1153 }

```

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

```
1154 \cs_new_protected:Npn \@@_pre_array_ii:
1155 {
```

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.

```
1156 \xglobal \colorlet { nicematrix } { . }
```

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

```
1157 \int_gzero:N \g_@@_total_X_weight_int
1158 \@@_expand_clist:N \l_@@_hlines_clist
1159 \@@_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 ⁶³.

```
1160 \bool_if:NT \c_@@_booktabs_loaded_bool
1161 { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1162 \box_clear_new:N \l_@@_cell_box
1163 \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}`).

```
1164 \bool_if:NT \l_@@_small_bool
1165 {
1166     \cs_set_nopar:Npn \arraystretch { 0.47 }
1167     \dim_set:Nn \arraycolsep { 1.45 pt }
1168 }

1169 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1170 {
1171     \tl_put_right:Nn \@@_begin_of_row:
1172     {
1173         \pgfsys@markposition
1174         { \@@_env: - row - \int_use:N \c@iRow - base }
1175     }
1176 }
```

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`.

```
1177 \cs_set_nopar:Npn \ialign
1178 {
1179     \bool_if:NTF \c_@@_colortbl_loaded_bool
1180     {
1181         \CT@everycr
1182         {
1183             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1184             \@@_everycr:

```

⁶³cf. `\nicematrix@redefine@check@rerun`


```

1185     }
1186   }
1187   { \everycr { \@@_everycr: } }
1188   \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`⁶⁴ 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`.

```

1189   \dim_gzero_new:N \g_@@_dp_row_zero_dim
1190   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1191   \dim_gzero_new:N \g_@@_ht_row_zero_dim
1192   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1193   \dim_gzero_new:N \g_@@_ht_row_one_dim
1194   \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1195   \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1196   \dim_gzero_new:N \g_@@_ht_last_row_dim
1197   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1198   \dim_gzero_new:N \g_@@_dp_last_row_dim
1199   \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`.

```

1200   \cs_set_eq:NN \ialign \@@_old_ialign:
1201   \halign
1202 }

```

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).

```

1203   \cs_set_eq:NN \@@_old_ldots \ldots
1204   \cs_set_eq:NN \@@_old_cdots \cdots
1205   \cs_set_eq:NN \@@_old_vdots \vdots
1206   \cs_set_eq:NN \@@_old_ddots \ddots
1207   \cs_set_eq:NN \@@_old_iddots \iddots
1208   \bool_if:NTF \l_@@_standard_cline_bool
1209   { \cs_set_eq:NN \cline \@@_standard_cline }
1210   { \cs_set_eq:NN \cline \@@_cline }
1211   \cs_set_eq:NN \Ldots \@@_Ldots
1212   \cs_set_eq:NN \Cdots \@@_Cdots
1213   \cs_set_eq:NN \Vdots \@@_Vdots
1214   \cs_set_eq:NN \Ddots \@@_Ddots
1215   \cs_set_eq:NN \Iddots \@@_Iddots
1216   \cs_set_eq:NN \hdottedline \@@_hdottedline:
1217   \cs_set_eq:NN \Hline \@@_Hline:
1218   \cs_set_eq:NN \Hspace \@@_Hspace:
1219   \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1220   \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1221   \cs_set_eq:NN \Block \@@_Block:
1222   \cs_set_eq:NN \rotate \@@_rotate:
1223   \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1224   \cs_set_eq:NN \dotfill \@@_old_dotfill:
1225   \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1226   \cs_set_eq:NN \diagbox \@@_diagbox:nn
1227   \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1228   \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1229   \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1230   \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:

```

⁶⁴The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

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. The command `\AtBeginEnvironment` is the command of `l3hooks` and, if this command is not available (versions of LaTeX prior to 2020-10-01), `etoolbox` is loaded and the command `\AtBeginDocument` of `etoolbox` is used.

```

1231 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1232 \AtBeginEnvironment { tabular }
1233 { \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).

```

1234 \seq_gclear:N \g_@@_multicolumn_cells_seq
1235 \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).

```

1236 \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`).

```

1237 \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.

```

1238 \int_gzero_new:N \g_@@_col_total_int
1239 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1240 \@@_renew_NC@rewrite@S:
1241 \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.

```

1242 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1243 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1244 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1245 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1246 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1247 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1248 \tl_gclear_new:N \g_nicematrix_code_before_tl
1249 }

```

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

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

```

1250 \cs_new_protected:Npn \@@_pre_array:
1251 {
1252 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1253 \int_gzero_new:N \c@iRow
1254 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1255 \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).

```

1256 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1257 {
1258   \bool_set_true:N \l_@@_last_row_without_value_bool
1259   \bool_if:NT \g_@@_aux_found_bool
1260     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \c_@@_size_seq 3 } }
1261 }
1262 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1263 {
1264   \bool_if:NT \g_@@_aux_found_bool
1265     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \c_@@_size_seq 6 } }
1266 }

```

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”.

```

1267 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1268 {
1269   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1270     {
1271       \dim_gset:Nn \g_@@_ht_last_row_dim
1272         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1273       \dim_gset:Nn \g_@@_dp_last_row_dim
1274         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1275     }
1276 }

1277 \seq_gclear:N \g_@@_cols_vlism_seq
1278 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1279 \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.

```

1280 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1281 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1282 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1283 \@@_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.

```

1284 \box_clear_new:N \l_@@_the_array_box

```

The preamble will be constructed in `\g_@@_preamble_tl`.

```

1285 \@@_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]`).

```

1286 \dim_zero_new:N \l_@@_left_delim_dim
1287 \dim_zero_new:N \l_@@_right_delim_dim
1288 \bool_if:NTF \l_@@_NiceArray_bool
1289 {
1290   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1291   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1292 }
1293 {

```

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

```

1294 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1295 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1296 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1297 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1298 }

```

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).

```

1299 \hbox_set:Nw \l_@@_the_array_box
1300 \skip_horizontal:N \l_@@_left_margin_dim
1301 \skip_horizontal:N \l_@@_extra_left_margin_dim
1302 \c_math_toggle_token
1303 \bool_if:NTF \l_@@_light_syntax_bool
1304 { \use:c { @@-light-syntax } }
1305 { \use:c { @@-normal-syntax } }
1306 }

```

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

```

1307 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1308 {
1309   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1310   \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...

```

1311 \@@_pre_array:
1312 }

```

The `\CodeBefore`

The following command will be executed if the `\CodeBefore` has to be actually executed.

```

1313 \cs_new_protected:Npn \@@_pre_code_before:
1314 {

```

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).

```

1315 \int_set:Nn \c@iRow { \seq_item:Nn \c_@@_size_seq 2 }
1316 \int_set:Nn \c@jCol { \seq_item:Nn \c_@@_size_seq 5 }
1317 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \c_@@_size_seq 3 }
1318 \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.

```

1319 \pgfsys@markposition { \@@_env: - position }
1320 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1321 \pgfpicture
1322 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1323 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1324 {
1325   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1326   \pgfcoordinate { \@@_env: - row - ##1 }
1327   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1328 }

```

Now, the recreation of the `col` nodes.

```

1329 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1330 {
1331   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1332   \pgfcoordinate { \@@_env: - col - ##1 }
1333   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1334 }

```

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

```

1335 \@@_create_diag_nodes:

```

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

```

1336 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1337 \endpgfpicture

```

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

```

1338 \@@_create_blocks_nodes:
1339 \bool_if:NT \c_@@_tikz_loaded_bool
1340 {
1341   \tikzset
1342   {
1343     every~picture / .style =
1344     { overlay , name~prefix = \@@_env: - }
1345   }
1346 }
1347 \cs_set_eq:NN \cellcolor \@@_cellcolor
1348 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1349 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1350 \cs_set_eq:NN \rowcolor \@@_rowcolor
1351 \cs_set_eq:NN \rowcolors \@@_rowcolors
1352 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1353 \cs_set_eq:NN \arraycolor \@@_arraycolor
1354 \cs_set_eq:NN \columncolor \@@_columncolor
1355 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1356 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1357 }

```

```

1358 \cs_new_protected:Npn \@@_exec_code_before:
1359 {
1360   \seq_gclear_new:N \g_@@_colors_seq
1361   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1362   \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`.

```

1363 \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`.

```
1364 \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.

```
1365 \@@_actually_color:
1366 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1367 \group_end:
1368 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1369 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1370 }
```

```
1371 \keys_define:nn { NiceMatrix / CodeBefore }
1372 {
1373   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1374   create-cell-nodes .default:n = true ,
1375   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1376   sub-matrix .value_required:n = true ,
1377   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1378   delimiters / color .value_required:n = true ,
1379   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1380 }

1381 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1382 {
1383   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1384   \@@_CodeBefore:w
1385 }
```

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.

```
1386 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1387 {
1388   \bool_if:NT \g_@@_aux_found_bool
1389   {
1390     \@@_pre_code_before:
1391     #1
1392   }
1393 }
```

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.

```
1394 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1395 {
1396   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1397   {
1398     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1399     \pgfcoordinate { \@@_env: - row - ##1 - base }
1400     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1401     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1402     {
1403       \cs_if_exist:cT
1404       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
```

```

1405         {
1406             \pgfsys@getposition
1407             { \@@_env: - ##1 - ####1 - NW }
1408             \@@_node_position:
1409             \pgfsys@getposition
1410             { \@@_env: - ##1 - ####1 - SE }
1411             \@@_node_position_i:
1412             \@@_pgf_rect_node:nnn
1413             { \@@_env: - ##1 - ####1 }
1414             { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1415             { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1416         }
1417     }
1418 }
1419 \int_step_inline:nn \c@iRow
1420 {
1421     \pgfnodealias
1422     { \@@_env: - ##1 - last }
1423     { \@@_env: - ##1 - \int_use:N \c@jCol }
1424 }
1425 \int_step_inline:nn \c@jCol
1426 {
1427     \pgfnodealias
1428     { \@@_env: - last - ##1 }
1429     { \@@_env: - \int_use:N \c@iRow - ##1 }
1430 }
1431 \@@_create_extra_nodes:
1432 }

```

```

1433 \cs_new_protected:Npn \@@_create_blocks_nodes:
1434 {
1435     \pgfpicture
1436     \pgf@relevantforpicturesizefalse
1437     \pgfrememberpicturepositiononpagetrue
1438     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1439     { \@@_create_one_block_node:nnnnn ##1 }
1440     \endpgfpicture
1441 }

```

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.⁶⁵

```

1442 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1443 {
1444     \tl_if_empty:nF { #5 }
1445     {
1446         \@@_qpoint:n { col - #2 }
1447         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1448         \@@_qpoint:n { #1 }
1449         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1450         \@@_qpoint:n { col - \@@_succ:n { #4 } }
1451         \dim_set_eq:NN \l_tmpc_dim \pgf@x
1452         \@@_qpoint:n { \@@_succ:n { #3 } }
1453         \dim_set_eq:NN \l_tmpd_dim \pgf@y
1454         \@@_pgf_rect_node:nnnnn
1455         { \@@_env: - #5 }
1456         { \dim_use:N \l_tmpa_dim }
1457         { \dim_use:N \l_tmpb_dim }
1458         { \dim_use:N \l_tmpc_dim }
1459         { \dim_use:N \l_tmpd_dim }

```

⁶⁵Moreover, 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).

```

1460   }
1461 }

1462 \cs_new_protected:Npn \@@_patch_for_revtext:
1463 {
1464   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1465   \cs_set_eq:NN \insert@column \insert@column@array
1466   \cs_set_eq:NN \@classx \@classx@array
1467   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1468   \cs_set_eq:NN \@arraycr \@arraycr@array
1469   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1470   \cs_set_eq:NN \array \array@array
1471   \cs_set_eq:NN \@array \@array@array
1472   \cs_set_eq:NN \@tabular \@tabular@array
1473   \cs_set_eq:NN \mkpream \mkpream@array
1474   \cs_set_eq:NN \endarray \endarray@array
1475   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1476   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1477 }

```

The environment {NiceArrayWithDelims}

```

1478 \NewDocumentEnvironment { NiceArrayWithDelims }
1479 { m m O { } m ! O { } t \CodeBefore }
1480 {
1481   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1482   \@@_provide_pgfsyspdfmark:
1483   \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.

```

1484   \bgroup

1485   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1486   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1487   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1488   \int_gzero:N \g_@@_block_box_int
1489   \dim_zero:N \g_@@_width_last_col_dim
1490   \dim_zero:N \g_@@_width_first_col_dim
1491   \bool_gset_false:N \g_@@_row_of_col_done_bool
1492   \str_if_empty:NT \g_@@_name_env_str
1493     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1494   \bool_if:NTF \l_@@_NiceTabular_bool
1495     \mode_leave_vertical:
1496     \@@_test_if_math_mode:
1497   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1498   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁶⁶. 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.

```

1499   \cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@

```

⁶⁶e.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.

```

1500   \cs_if_exist:NT \tikz@library@external@loaded
1501   {
1502     \tikzexternaldisable
1503     \cs_if_exist:NT \ifstandalone
1504     { \tikzset { external / optimize = false } }
1505   }

```

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

```

1506   \int_gincr:N \g_@@_env_int
1507   \bool_if:NF \l_@@_block_auto_columns_width_bool
1508   { \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`).

```

1509   \seq_gclear:N \g_@@_blocks_seq
1510   \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`.

```

1511   \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1512   \seq_gclear:N \g_@@_pos_of_xdots_seq
1513   \tl_gclear_new:N \g_@@_code_before_tl
1514   \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.

```

1515   \bool_gset_false:N \g_@@_aux_found_bool
1516   \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1517   {
1518     \bool_gset_true:N \g_@@_aux_found_bool
1519     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1520   }

```

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.

```

1521   \tl_gclear:N \g_@@_aux_tl
1522   \tl_if_empty:NF \g_@@_code_before_tl
1523   {
1524     \bool_set_true:N \l_@@_code_before_bool
1525     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1526   }

```

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`.

```

1527   \bool_if:NTF \l_@@_NiceArray_bool
1528   { \keys_set:nn { NiceMatrix / NiceArray } }
1529   { \keys_set:nn { NiceMatrix / pNiceArray } }
1530   { #3 , #5 }

1531   \tl_if_empty:NF \l_@@_rules_color_tl
1532   { \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:.`

```

1533   \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:

```

```

1534 }
1535 {
1536   \bool_if:NTF \l_@@_light_syntax_bool
1537     { \use:c { end @@-light-syntax } }
1538     { \use:c { end @@-normal-syntax } }
1539   \c_math_toggle_token
1540   \skip_horizontal:N \l_@@_right_margin_dim
1541   \skip_horizontal:N \l_@@_extra_right_margin_dim
1542   \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.

```

1543   \bool_if:NT \l_@@_width_used_bool
1544   {
1545     \int_compare:nNnT \g_@@_total_X_weight_int = 0
1546       { \@@_error:n { width~without~X~columns } }
1547   }

```

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 .

```

1548   \int_compare:nNnT \g_@@_total_X_weight_int > 0
1549   {
1550     \tl_gput_right:Nx \g_@@_aux_tl
1551     {
1552       \bool_set_true:N \l_@@_X_columns_aux_bool
1553       \dim_set:Nn \l_@@_X_columns_dim
1554       {
1555         \dim_compare:nNnTF
1556           {
1557             \dim_abs:n
1558             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1559           }
1560           <
1561           { 0.001 pt }
1562           { \dim_use:N \l_@@_X_columns_dim }
1563           {
1564             \dim_eval:n
1565             {
1566               ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1567               / \int_use:N \g_@@_total_X_weight_int
1568               + \l_@@_X_columns_dim
1569             }
1570           }
1571       }
1572     }
1573   }

```

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).

```

1574   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1575   {
1576     \bool_if:NF \l_@@_last_row_without_value_bool
1577     {
1578       \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1579       {
1580         \@@_error:n { Wrong~last~row }
1581         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1582       }
1583     }
1584   }

```

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”.⁶⁷

```

1585 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1586 \bool_if:nTF \g_@@_last_col_found_bool
1587 { \int_gdecr:N \c@jCol }
1588 {
1589   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1590   { \@@_error:n { last~col~not~used } }
1591 }

```

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

```

1592 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1593 \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. 126).

```

1594 \int_compare:nNnT \l_@@_first_col_int = 0
1595 {
1596   \skip_horizontal:N \col@sep
1597   \skip_horizontal:N \g_@@_width_first_col_dim
1598 }

```

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.

```

1599 \bool_if:nTF \l_@@_NiceArray_bool
1600 {
1601   \str_case:VnF \l_@@_baseline_tl
1602   {
1603     b \@@_use_arraybox_with_notes_b:
1604     c \@@_use_arraybox_with_notes_c:
1605   }
1606   \@@_use_arraybox_with_notes:
1607 }

```

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).

```

1608 {
1609   \int_compare:nNnTF \l_@@_first_row_int = 0
1610   {
1611     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1612     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1613   }
1614   { \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”.⁶⁸

```

1615 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1616 {
1617   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1618   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1619 }
1620 { \dim_zero:N \l_tmpb_dim }
1621 \hbox_set:Nn \l_tmpa_box
1622 {
1623   \c_math_toggle_token
1624   \tl_if_empty:NF \l_@@_delimiters_color_tl
1625   { \color { \l_@@_delimiters_color_tl } }

```

⁶⁷We remind that the potential “first column” (exterior) has the number 0.

⁶⁸A 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).

```

1626         \exp_after:wN \left \g_@@_left_delim_tl
1627         \vcenter
1628         {

```

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).

```

1629         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1630         \hbox
1631         {
1632             \bool_if:NTF \l_@@_NiceTabular_bool
1633             { \skip_horizontal:N -\tabcolsep }
1634             { \skip_horizontal:N -\arraycolsep }
1635             \@@_use_arraybox_with_notes_c:
1636             \bool_if:NTF \l_@@_NiceTabular_bool
1637             { \skip_horizontal:N -\tabcolsep }
1638             { \skip_horizontal:N -\arraycolsep }
1639         }

```

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).

```

1640         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1641     }

```

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.

```

1642         \tl_if_empty:NF \l_@@_delimiters_color_tl
1643         { \color { \l_@@_delimiters_color_tl } }
1644         \exp_after:wN \right \g_@@_right_delim_tl
1645         \c_math_toggle_token
1646     }

```

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.

```

1647         \bool_if:NTF \l_@@_delimiters_max_width_bool
1648         {
1649             \@@_put_box_in_flow_bis:nn
1650             \g_@@_left_delim_tl \g_@@_right_delim_tl
1651         }
1652         \@@_put_box_in_flow:
1653     }

```

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. 127).

```

1654         \bool_if:NT \g_@@_last_col_found_bool
1655         {
1656             \skip_horizontal:N \g_@@_width_last_col_dim
1657             \skip_horizontal:N \col@sep
1658         }
1659         \bool_if:NF \l_@@_Matrix_bool
1660         {
1661             \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1662             { \@@_error:n { columns-not-used } }
1663         }
1664         \group_begin:
1665         \globaldefs = 1
1666         \@@_msg_redirect_name:nn { columns-not-used } { error }
1667         \group_end:
1668         \@@_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.

```

1669         \egroup

```

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

```

1670 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1671 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1672 \iow_now:Nx \@mainaux
1673 {
1674   \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _tl }
1675   { \exp_not:V \g_@@_aux_tl }
1676 }
1677 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1678 \bool_if:NT \c_@@_footnote_bool \endsavenotes
1679 }

```

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.

```

1680 \cs_new_protected:Npn \@@_construct_preamble:
1681 {

```

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 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.

```

1682   \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).

```

1683   \bool_if:NF \l_@@_Matrix_bool
1684   {
1685     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1686     \@@_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).

```

1687   \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).

```

1688   \exp_args:NV \@temptokena \g_@@_preamble_tl

```

Initialisation of a flag used by `array` to detect the end of the expansion.

```

1689   \@tempswatrue

```

The following line actually does the expansion (it’s has been copied from `array.sty`). The expanded version is still in `\@temptokena`.

```

1690   \@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`.

```

1691      \int_gzero:N \c@jCol
1692      \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1693      \bool_gset_false:N \g_tmpb_bool
1694      \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1695      {
1696          \tl_gset:Nn \g_@@_preamble_tl
1697              { ! { \skip_horizontal:N \arrayrulewidth } }
1698      }
1699      {
1700          \clist_if_in:NnT \l_@@_vlines_clist 1
1701          {
1702              \tl_gset:Nn \g_@@_preamble_tl
1703                  { ! { \skip_horizontal:N \arrayrulewidth } }
1704          }
1705      }

```

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`).

```

1706      \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1707      \int_zero:N \l_tmpa_int

```

Now, we actually patch the preamble (and it is constructed in `\g_@@_preamble_tl`).

```

1708      \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1709      \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1710  }

```

Now, we replace `\columncolor` by `\@@_columncolor_preamble`.

```

1711      \bool_if:NT \l_@@_colortbl_like_bool
1712      {
1713          \regex_replace_all:NnN
1714              \c_@@_columncolor_regex
1715              { \c { @@_columncolor_preamble } }
1716              \g_@@_preamble_tl
1717      }

```

Now, we can close the TeX group which was opened for the redefinition of the columns of type `w` and `W`.

```

1718      \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}`.

```

1719      \bool_lazy_or:nnT
1720          { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1721          { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1722      { \bool_set_false:N \l_@@_NiceArray_bool }

```

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

```

1723      \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).

```

1724      \int_compare:nNnTF \l_@@_first_col_int = 0
1725      { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1726      {
1727          \bool_lazy_all:nT

```

```

1728     {
1729         \l_@@_NiceArray_bool
1730         { \bool_not_p:n \l_@@_NiceTabular_bool }
1731         { \tl_if_empty_p:N \l_@@_vlines_clist }
1732         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1733     }
1734     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1735 }
1736 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1737 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1738 {
1739     \bool_lazy_all:nT
1740     {
1741         \l_@@_NiceArray_bool
1742         { \bool_not_p:n \l_@@_NiceTabular_bool }
1743         { \tl_if_empty_p:N \l_@@_vlines_clist }
1744         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1745     }
1746     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1747 }

```

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`).

```

1748     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1749     {
1750         \tl_gput_right:Nn \g_@@_preamble_tl
1751         { > { \@@_error_too_much_cols: } 1 }
1752     }
1753 }

```

```

1754 \cs_new_protected:Npn \@@_patch_preamble:n #1
1755 {
1756     \str_case:nnF { #1 }
1757     {
1758         c { \@@_patch_preamble_i:n #1 }
1759         l { \@@_patch_preamble_i:n #1 }
1760         r { \@@_patch_preamble_i:n #1 }
1761         > { \@@_patch_preamble_ii:nn #1 }
1762         ! { \@@_patch_preamble_ii:nn #1 }
1763         @ { \@@_patch_preamble_ii:nn #1 }
1764         | { \@@_patch_preamble_iii:n #1 }
1765         p { \@@_patch_preamble_iv:n #1 }
1766         b { \@@_patch_preamble_iv:n #1 }
1767         m { \@@_patch_preamble_iv:n #1 }
1768         \@@_V: { \@@_patch_preamble_v:n }
1769         V { \@@_patch_preamble_v:n }
1770         \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
1771         \@@_W: { \@@_patch_preamble_vi:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1772         \@@_S: { \@@_patch_preamble_vii:n }
1773         ( { \@@_patch_preamble_viii:nn #1 }
1774         [ { \@@_patch_preamble_viii:nn #1 }
1775         \{ { \@@_patch_preamble_viii:nn #1 }
1776         ) { \@@_patch_preamble_ix:nn #1 }
1777         ] { \@@_patch_preamble_ix:nn #1 }
1778         \} { \@@_patch_preamble_ix:nn #1 }
1779         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’.

```

1780     \@@_X { \@@_patch_preamble_x:n }
1781     \q_stop { }
1782 }

```

```

1783 {
1784   \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1785   { \@@_patch_preamble_xii:n #1 }
1786   {
1787     \str_if_eq:VnTF \l_@@_letter_vlism_tl { #1 }
1788     {
1789       \seq_gput_right:Nx \g_@@_cols_vlism_seq
1790       { \int_eval:n { \c@jCol + 1 } }
1791       \tl_gput_right:Nx \g_@@_preamble_tl
1792       { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1793       \@@_patch_preamble:n
1794     }
1795     {
1796       \bool_lazy_and:nnTF
1797       { \str_if_eq_p:nn { : } { #1 } }
1798       \c_@@_arydshln_loaded_bool
1799       {
1800         \tl_gput_right:Nn \g_@@_preamble_tl { : }
1801         \@@_patch_preamble:n
1802       }
1803       { \@@_fatal:nn { unknown-column-type } { #1 } }
1804     }
1805   }
1806 }
1807 }

```

For c, l and r

```

1808 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1809 {
1810   \tl_gput_right:Nn \g_@@_preamble_tl
1811   {
1812     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
1813     #1
1814     < \@@_cell_end:
1815   }

```

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

```

1816   \int_gincr:N \c@jCol
1817   \@@_patch_preamble_xi:n
1818 }

```

For >, ! and @

```

1819 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1820 {
1821   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1822   \@@_patch_preamble:n
1823 }

```

For |

```

1824 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1825 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1826   \int_incr:N \l_tmpa_int
1827   \@@_patch_preamble_iii_i:n
1828 }

```

```

1829 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1830 {
1831   \str_if_eq:nnTF { #1 } |
1832   { \@@_patch_preamble_iii:n | }
1833   {
1834     \tl_gput_right:Nx \g_@@_preamble_tl
1835     {

```



```

1836         \exp_not:N !
1837         {
1838             \skip_horizontal:n
1839             {
1840                 \dim_eval:n
1841                 {
1842                     \arrayrulewidth * \l_tmpa_int
1843                     + \doublerulesep * ( \l_tmpa_int - 1)
1844                 }
1845             }
1846         }
1847     }
1848     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1849     {
1850         \@Q_vline:nnnn
1851         { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } { 1 } { }
1852     }
1853     \int_zero:N \l_tmpa_int
1854     \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
1855     \@@_patch_preamble:n #1
1856 }
1857 }
1858 \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.

```

1859 \keys_define:nn { WithArrows / p-column }
1860 {
1861     r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
1862     r .value_forbidden:n = true ,
1863     c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
1864     c .value_forbidden:n = true ,
1865     l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
1866     l .value_forbidden:n = true ,
1867     si .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
1868     si .value_forbidden:n = true ,
1869     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
1870     p .value_forbidden:n = true ,
1871     t .meta:n = p ,
1872     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
1873     m .value_forbidden:n = true ,
1874     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
1875     b .value_forbidden:n = true ,
1876 }

```

For `p`, `b` and `m`. The argument `#1` is that value : `p`, `b` or `m`.

```

1877 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
1878 {
1879     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

1880     \@@_patch_preamble_iv_i:n
1881 }
1882 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
1883 {
1884     \str_if_eq:nnTF { #1 } { [ ]
1885         { \@@_patch_preamble_iv_ii:w [ ]
1886           { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
1887         }
1888     \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
1889     { \@@_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.

```
1890 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
1891 {
```

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).

```
1892   \str_set:Nn \l_@@_hpos_col_str { j }
1893   \keys_set:nn { WithArrows / p-column } { #1 }
1894   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
1895 }
```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`.

```
1896 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
1897 {
1898   \use:x
1899   {
1900     \@@_patch_preamble_iv_v:nnnnnnnn
1901     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
1902     { \dim_eval:n { #1 } }
1903   }
```

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.

```
1904       \str_if_eq:VnTF \l_@@_hpos_col_str j
1905       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
1906       {
1907         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
1908         { \l_@@_hpos_col_str }
1909       }
1910     \str_case:Vn \l_@@_hpos_col_str
1911     {
1912       c { \exp_not:N \centering }
1913       l { \exp_not:N \raggedright }
1914       r { \exp_not:N \raggedleft }
1915     }
1916   }
1917   { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
1918   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
1919   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
1920   { #2 }
1921   {
1922     \str_case:VnF \l_@@_hpos_col_str
1923     {
1924       { j } { c }
1925       { si } { c }
1926     }
1927     { \l_@@_hpos_col_str }
1928   }
1929 }
```

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

```
1930   \int_gincr:N \c@jCol
1931   \@@_patch_preamble_xi:n
1932 }
```

#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*.

```

1933 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
1934 {
1935   \tl_gput_right:Nn \g_@@_preamble_tl
1936   {
1937     > {

```

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.

```

1938       \dim_set:Nn \l_@@_col_width_dim { #2 }
1939       \@@_cell_begin:w
1940       \begin { #7 } [ #1 ] { #2 }

```

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

```

1941       \everypar
1942       {
1943         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
1944         \everypar { }
1945       }

```

Now, the potential code for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing).

```

1946       #3

```

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

```

1947       \g_@@_row_style_tl
1948       \arraybackslash
1949       #5
1950     }
1951     #8
1952     < {
1953       #6

```

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

```

1954       \@finalstrut \@arstrutbox
1955       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
1956       \end { #7 }

```

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

```

1957       #4
1958       \@@_cell_end:
1959     }
1960   }
1961 }

```

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

```

1962 \cs_new_protected:Npn \@@_center_cell_box:
1963 {

```

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

```

1964 \tl_gput_right:Nn \g_@@_post_action_cell_tl
1965 {
1966   \int_compare:nNnT
1967     { \box_ht:N \l_@@_cell_box }
1968     >
1969     { \box_ht:N \@arstrutbox }
1970     {
1971       \hbox_set:Nn \l_@@_cell_box
1972         {
1973           \box_move_down:nn
1974             {
1975               ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
1976                 + \baselineskip ) / 2
1977             }
1978           { \box_use:N \l_@@_cell_box }
1979         }
1980     }
1981 }
1982 }
```

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

```

1983 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
1984 {
1985   \str_if_eq:nnTF { #1 } { [ ] }
1986     { \@@_patch_preamble_v_i:w [ ] }
1987     { \@@_patch_preamble_v_i:w [ ] { #1 } }
1988 }
1989 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
1990 { \@@_patch_preamble_v_ii:nn { #1 } }
1991 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
1992 {
1993   \str_set:Nn \l_@@_vpos_col_str { p }
1994   \str_set:Nn \l_@@_hpos_col_str { j }
1995   \keys_set:nn { WithArrows / p-column } { #1 }
1996   \bool_if:NTF \c_@@_varwidth_loaded_bool
1997     { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
1998     {
1999       \@@_error:n { varwidth~not~loaded }
2000       \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2001     }
2002 }
```

For `w` and `W`

```

2003 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2004 {
2005   \tl_gput_right:Nn \g_@@_preamble_tl
2006     {
2007       > {
```

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.

```

2008       \dim_set:Nn \l_@@_col_width_dim { #4 }
2009       \hbox_set:Nw \l_@@_cell_box
2010       \@@_cell_begin:w
2011       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2012     }
2013   c
2014   < {
2015     \@@_cell_end:
2016     #1
2017     \hbox_set_end:
```

```

2018         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2019         \@@_adjust_size_box:
2020         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2021     }
2022 }

```

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

```

2023     \int_gincr:N \c@jCol
2024     \@@_patch_preamble_xi:n
2025 }

```

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).

```

2026 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2027 {
2028     \str_if_eq:nnTF { #1 } { [ ]
2029     { \@@_patch_preamble_vii_i:w [ ]
2030     { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2031 }
2032 \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2033 { \@@_patch_preamble_vii_ii:n { #1 } }
2034 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2035 {

```

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.

```

2036     \cs_if_exist:NTF \siunitx_cell_begin:w
2037     {
2038         \tl_gput_right:Nn \g_@@_preamble_tl
2039         {
2040             > {
2041                 \@@_cell_begin:w
2042                 \keys_set:nn { siunitx } { #1 }
2043                 \siunitx_cell_begin:w
2044             }
2045             c
2046             < { \siunitx_cell_end: \@@_cell_end: }
2047         }

```

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

```

2048     \int_gincr:N \c@jCol
2049     \@@_patch_preamble_xi:n
2050 }
2051 { \@@_fatal:n { Version~of~siunitx~too~old } }
2052 }

```

For (, [and \{.

```

2053 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2054 {
2055     \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.

```

2056     \int_compare:nNnTF \c@jCol = \c_zero_int
2057     {
2058         \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2059         {

```

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

```

2060         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2061         \tl_gset:Nn \g_@@_right_delim_tl { . }
2062         \@@_patch_preamble:n #2
2063     }
2064 {

```

```

2065         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2066         \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2067     }
2068 }
2069 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2070 }
2071 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2072 {
2073     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2074     { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
2075     \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
2076     {
2077         \@@_error:nn { delimiter~after~opening } { #2 }
2078         \@@_patch_preamble:n
2079     }
2080     { \@@_patch_preamble:n #2 }
2081 }

```

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}`).

```

2082 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2083 {
2084     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2085     \tl_if_in:nnTF { ) ] \} } { #2 }
2086     { \@@_patch_preamble_ix_i:nnn #1 #2 }
2087     {
2088         \tl_if_eq:nnTF { \q_stop } { #2 }
2089         {
2090             \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2091             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2092             {
2093                 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2094                 \tl_gput_right:Nx \g_@@_internal_code_after_tl
2095                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2096                 \@@_patch_preamble:n #2
2097             }
2098         }
2099         {
2100             \tl_if_in:nnT { ( [ \{ } { #2 }
2101             { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2102             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2103             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2104             \@@_patch_preamble:n #2
2105         }
2106     }
2107 }
2108 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2109 {
2110     \tl_if_eq:nnTF { \q_stop } { #3 }
2111     {
2112         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2113         {
2114             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2115             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2116             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2117             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2118         }
2119         {
2120             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2121             \tl_gput_right:Nx \g_@@_internal_code_after_tl

```

```

2122         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2123         \@@_error:nn { double-closing-delimiter } { #2 }
2124     }
2125 }
2126 {
2127     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2128     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2129     \@@_error:nn { double-closing-delimiter } { #2 }
2130     \@@_patch_preamble:n #3
2131 }
2132 }

```

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.

```

2133 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2134 {
2135     \str_if_eq:nnTF { #1 } { [ ]
2136         { \@@_patch_preamble_x_i:w [ ]
2137           { \@@_patch_preamble_x_i:w [ ] #1 }
2138         }
2139     \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2140     { \@@_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).

```

2141 \keys_define:nn { WithArrows / X-column }
2142 { 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.

```

2143 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2144 {

```

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).

```

2145     \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).

```

2146     \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.

```

2147     \int_zero_new:N \l_@@_weight_int
2148     \int_set:Nn \l_@@_weight_int { 1 }
2149     \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
2150     \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2151     \int_compare:nNnT \l_@@_weight_int < 0
2152     {
2153         \exp_args:Nnx \@@_error:nn { negative-weight }
2154         { \int_use:N \l_@@_weight_int }
2155         \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2156     }
2157     \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).

```

2158     \bool_if:NTF \l_@@_X_columns_aux_bool
2159     {
2160         \@@_patch_preamble_iv_iv:nn

```

```

2161     { \l_@@_weight_int \l_@@_X_columns_dim }
2162     { minipage }
2163   }
2164   {
2165     \tl_gput_right:Nn \g_@@_preamble_tl
2166     {
2167       > {
2168         \@@_cell_begin:w
2169         \bool_set_true:N \l_@@_X_column_bool

```

The following code will nullify the box of the cell.

```

2170     \tl_gput_right:Nn \g_@@_post_action_cell_tl
2171     { \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`.

```

2172     \begin { minipage } { 5 cm } \arraybackslash
2173   }
2174   c
2175   < {
2176     \end { minipage }
2177     \@@_cell_end:
2178   }
2179 }
2180 \int_gincr:N \c@jCol
2181 \@@_patch_preamble_xi:n
2182 }
2183 }

```

```

2184 \cs_new_protected:Npn \@@_patch_preamble_xii:n #1
2185 {
2186   \tl_gput_right:Nn \g_@@_preamble_tl
2187   { ! { \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`.

```

2188   \tl_gput_right:Nx \g_@@_internal_code_after_tl
2189   { \@@_vdottedline:n { \int_use:N \c@jCol } }
2190   \@@_patch_preamble:n
2191 }

```

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.

```

2192 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2193 {
2194   \str_if_eq:nnTF { #1 } { < }
2195   \@@_patch_preamble_xiii:n
2196   {
2197     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2198     {
2199       \tl_gput_right:Nn \g_@@_preamble_tl
2200       { ! { \skip_horizontal:N \arrayrulewidth } }
2201     }
2202     {
2203       \exp_args:NNx
2204       \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2205       {
2206         \tl_gput_right:Nn \g_@@_preamble_tl
2207         { ! { \skip_horizontal:N \arrayrulewidth } }
2208       }
2209     }
2210     \@@_patch_preamble:n { #1 }
2211   }
2212 }

```



```

2213 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2214 {
2215     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2216     \@@_patch_preamble_xi:n
2217 }

```

The redefinition of `\multicolumn`

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

```

2218 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2219 {

```

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`.

```

2220     \multispan { #1 }
2221     \begingroup
2222     \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

You do the expansion of the (small) preamble with the tools of `array`.

```

2223     \@temptokena = { #2 }
2224     \@tempswatrue
2225     \@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.

```

2226     \tl_gclear:N \g_@@_preamble_tl
2227     \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

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

```

2228     \exp_args:NV \mkpream \g_@@_preamble_tl
2229     \@addtopreamble \@empty
2230     \endgroup

```

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

```

2231     \int_compare:nNnT { #1 } > 1
2232     {
2233         \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2234         { \int_use:N \c@iRow - \@@_succ:n \c@jCol }
2235         \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2236         \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2237         {
2238             { \int_use:N \c@iRow }
2239             { \int_eval:n { \c@jCol + 1 } } }
2240             { \int_use:N \c@iRow }
2241             { \int_eval:n { \c@jCol + #1 } }
2242             { } % for the name of the block
2243         }
2244     }

```

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

```

2245     \cs_set:Npn \@sharp { #3 }
2246     \@arstrut
2247     \@preamble
2248     \null

```

We add some lines.

```

2249 \int_gadd:Nn \c@jCol { #1 - 1 }
2250 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2251 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2252 \ignorespaces
2253 }

```

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`.

```

2254 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2255 {
2256   \str_case:nnF { #1 }
2257   {
2258     c { \@@_patch_m_preamble_i:n #1 }
2259     l { \@@_patch_m_preamble_i:n #1 }
2260     r { \@@_patch_m_preamble_i:n #1 }
2261     > { \@@_patch_m_preamble_ii:nn #1 }
2262     ! { \@@_patch_m_preamble_ii:nn #1 }
2263     @ { \@@_patch_m_preamble_ii:nn #1 }
2264     | { \@@_patch_m_preamble_iii:n #1 }
2265     p { \@@_patch_m_preamble_iv:nnn t #1 }
2266     m { \@@_patch_m_preamble_iv:nnn c #1 }
2267     b { \@@_patch_m_preamble_iv:nnn b #1 }
2268     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2269     \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2270     \q_stop { }
2271   }
2272   { \@@_fatal:nn { unknown~column~type } { #1 } }
2273 }

```

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

```

2274 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2275 {
2276   \tl_gput_right:Nn \g_@@_preamble_tl
2277   {
2278     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2279     #1
2280     < \@@_cell_end:
2281   }

```

We test for the presence of a `<`.

```

2282 \@@_patch_m_preamble_x:n
2283 }

```

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

```

2284 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2285 {
2286   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2287   \@@_patch_m_preamble:n
2288 }

```

For `|`

```

2289 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2290 {
2291   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2292   \@@_patch_m_preamble:n
2293 }

```

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

```

2294 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2295 {
2296   \tl_gput_right:Nn \g_@@_preamble_tl

```

```

2297 {
2298   > {
2299     \@@_cell_begin:w
2300     \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2301     \mode_leave_vertical:
2302     \arraybackslash
2303     \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2304   }
2305   c
2306   < {
2307     \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2308     \end { minipage }
2309     \@@_cell_end:
2310   }
2311 }

```

We test for the presence of a <.

```

2312 \@@_patch_m_preamble_x:n
2313 }

```

For w and W

```

2314 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2315 {
2316   \tl_gput_right:Nn \g_@@_preamble_tl
2317   {
2318     > {
2319       \hbox_set:Nw \l_@@_cell_box
2320       \@@_cell_begin:w
2321       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2322     }
2323     c
2324     < {
2325       \@@_cell_end:
2326       #1
2327       \hbox_set_end:
2328       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2329       \@@_adjust_size_box:
2330       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2331     }
2332   }

```

We test for the presence of a <.

```

2333 \@@_patch_m_preamble_x:n
2334 }

```

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 vl_lines is used.

```

2335 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2336 {
2337   \str_if_eq:nnTF { #1 } { < }
2338   \@@_patch_m_preamble_ix:n
2339   {
2340     \tl_if_eq:NnTF \l_@@_vl_lines_clist { all }
2341     {
2342       \tl_gput_right:Nn \g_@@_preamble_tl
2343       { ! { \skip_horizontal:N \arrayrulewidth } }
2344     }
2345     {
2346       \exp_args:NNx
2347       \clist_if_in:NnT \l_@@_vl_lines_clist { \@@_succ:n \c@jCol }
2348       {
2349         \tl_gput_right:Nn \g_@@_preamble_tl
2350         { ! { \skip_horizontal:N \arrayrulewidth } }
2351       }
2352     }

```

```

2353     \@@_patch_m_preamble:n { #1 }
2354   }
2355 }
2356 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2357 {
2358   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2359   \@@_patch_m_preamble_x:n
2360 }

```

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`).

```

2361 \cs_new_protected:Npn \@@_put_box_in_flow:
2362 {
2363   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2364   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2365   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2366     { \box_use_drop:N \l_tmpa_box }
2367   \@@_put_box_in_flow_i:
2368 }

```

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).

```

2369 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2370 {
2371   \pgfpicture
2372     \@@_qpoint:n { row - 1 }
2373     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2374     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
2375     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2376     \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).

```

2377   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2378   {
2379     \int_set:Nn \l_tmpa_int
2380     {
2381       \str_range:Nnn
2382         \l_@@_baseline_tl
2383         6
2384         { \tl_count:V \l_@@_baseline_tl }
2385     }
2386     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2387   }
2388   {
2389     \str_case:VnF \l_@@_baseline_tl
2390     {
2391       { t } { \int_set:Nn \l_tmpa_int 1 }
2392       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2393     }
2394     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2395     \bool_lazy_or:nnT
2396       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2397       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2398     {
2399       \@@_error:n { bad~value~for~baseline }
2400       \int_set:Nn \l_tmpa_int 1
2401     }
2402     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2403         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2404     }
2405     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2406     \endpgfpicture
2407     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2408     \box_use_drop:N \l_tmpa_box
2409 }

```

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).

```

2410 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2411 {

```

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.

```

2412     \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2413     {
2414         \box_set_wd:Nn \l_@@_the_array_box
2415         { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2416     }

```

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

```

2417     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }

```

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

```

2418     \hbox
2419     {
2420         \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.

```

2421         \@@_create_extra_nodes:
2422         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2423     }
2424     \bool_lazy_or:nnT
2425     { \int_compare_p:nNn \c@tabularnote > 0 }
2426     { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2427     \@@_insert_tabularnotes:
2428     \end { minipage }
2429 }

2430 \cs_new_protected:Npn \@@_insert_tabularnotes:
2431 {
2432     \skip_vertical:N 0.65ex

```

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

```

2433     \group_begin:
2434     \l_@@_notes_code_before_tl
2435     \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.

```

2436     \int_compare:nNnT \c@tabularnote > 0
2437     {
2438         \bool_if:NTF \l_@@_notes_para_bool
2439         {

```

```

2440         \begin { tabularnotes* }
2441         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2442         \end { tabularnotes* }

```

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

```

2443         \par
2444     }
2445     {
2446         \tabularnotes
2447         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2448         \endtabularnotes
2449     }
2450 }
2451 \unskip
2452 \group_end:
2453 \bool_if:NT \l_@@_notes_bottomrule_bool
2454 {
2455     \bool_if:NTF \c_@@_booktabs_loaded_bool
2456     {

```

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

```

2457         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2458         { \CT@arc@ \hrule height \heavyrulewidth }
2459     }
2460     { \@@_error:n { bottomrule~without~booktabs } }
2461 }
2462 \l_@@_notes_code_after_tl
2463 \seq_gclear:N \g_@@_tabularnotes_seq
2464 \int_gzero:N \c@tabularnote
2465 }

```

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`.

```

2466 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2467 {
2468     \pgfpicture
2469     \@@_qpoint:n { row - 1 }
2470     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2471     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2472     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2473     \endpgfpicture
2474     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2475     \int_compare:nNnT \l_@@_first_row_int = 0
2476     {
2477         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2478         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2479     }
2480     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2481 }

```

Now, the general case.

```

2482 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2483 {

```

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

```

2484     \tl_if_eq:NnT \l_@@_baseline_tl { t }
2485     { \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`.

```

2486 \pgfpicture
2487 \@@_qpoint:n { row - 1 }
2488 \dim_gset_eq:NW \g_tmpa_dim \pgf@y
2489 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2490 {
2491   \int_set:Nn \l_tmpa_int
2492   {
2493     \str_range:Nnn
2494       \l_@@_baseline_tl
2495       6
2496     { \tl_count:V \l_@@_baseline_tl }
2497   }
2498   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2499 }
2500 {
2501   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2502   \bool_lazy_or:nnT
2503     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2504     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2505   {
2506     \@@_error:n { bad-value-for-baseline }
2507     \int_set:Nn \l_tmpa_int 1
2508   }
2509   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2510 }
2511 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2512 \endpgfpicture
2513 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2514 \int_compare:nNnT \l_@@_first_row_int = 0
2515 {
2516   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2517   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2518 }
2519 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2520 }

```

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.

```

2521 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2522 {

```

We will compute the real width of both delimiters used.

```

2523 \dim_zero_new:N \l_@@_real_left_delim_dim
2524 \dim_zero_new:N \l_@@_real_right_delim_dim
2525 \hbox_set:Nn \l_tmpb_box
2526 {
2527   \c_math_toggle_token
2528   \left #1
2529   \vcenter
2530   {
2531     \vbox_to_ht:nn
2532       { \box_ht_plus_dp:N \l_tmpa_box }
2533       { }
2534   }
2535   \right .
2536   \c_math_toggle_token
2537 }
2538 \dim_set:Nn \l_@@_real_left_delim_dim
2539 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
2540 \hbox_set:Nn \l_tmpb_box

```

```

2541 {
2542   \c_math_toggle_token
2543   \left .
2544   \vbox_to_ht:nn
2545     { \box_ht_plus_dp:N \l_tmpa_box }
2546     { }
2547   \right #2
2548   \c_math_toggle_token
2549 }
2550 \dim_set:Nn \l_@@_real_right_delim_dim
2551 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2552 \skip_horizontal:N \l_@@_left_delim_dim
2553 \skip_horizontal:N -\l_@@_real_left_delim_dim
2554 \@@_put_box_in_flow:
2555 \skip_horizontal:N \l_@@_right_delim_dim
2556 \skip_horizontal:N -\l_@@_real_right_delim_dim
2557 }

```

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).

```

2558 \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).

```

2559 {
2560   \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`).

```

2561 { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2562 }
2563 {
2564   \@@_create_col_nodes:
2565   \endarray
2566 }

```

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`).

```

2567 \NewDocumentEnvironment { @@-light-syntax } { b }
2568 {

```

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`.

```

2569 \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
2570 \tl_map_inline:nn { #1 }
2571 {
2572   \str_if_eq:nnT { ##1 } { & }
2573   { \@@_fatal:n { ampersand-in~light-syntax } }
2574   \str_if_eq:nnT { ##1 } { \ }
2575   { \@@_fatal:n { double-backslash-in~light-syntax } }
2576 }

```


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`.

```
2577 \@@_light_syntax_i #1 \CodeAfter \q_stop
2578 }
```

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`.

```
2579 { }
2580 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2581 {
2582   \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).

```
2583 \seq_gclear_new:N \g_@@_rows_seq
2584 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2585 \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.

```
2586 \int_compare:nNnT \l_@@_last_row_int = { -1 }
2587 { \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`).

```
2588 \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.

```
2589 \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2590 \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
2591 \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2592 \@@_create_col_nodes:
2593 \endarray
2594 }
2595 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2596 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
2597 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2598 {
2599   \seq_gclear_new:N \g_@@_cells_seq
2600   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2601   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2602   \l_tmpa_tl
2603   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2604 }
```

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).

```
2605 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2606 {
2607   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2608   { \@@_fatal:n { empty-environment } } }
```

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

```
2609 \end { #2 }
2610 }
```

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).

```

2611 \cs_new:Npn \@@_create_col_nodes:
2612 {
2613   \crrc
2614   \int_compare:nNnT \l_@@_first_col_int = 0
2615   {
2616     \omit
2617     \hbox_overlap_left:n
2618     {
2619       \bool_if:NT \l_@@_code_before_bool
2620       { \pgfsys@markposition { \@@_env: - col - 0 } }
2621       \pgfpicture
2622       \pgfrememberpicturerepositiononpagetrue
2623       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
2624       \str_if_empty:NF \l_@@_name_str
2625       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2626       \endpgfpicture
2627       \skip_horizontal:N 2\col@sep
2628       \skip_horizontal:N \g_@@_width_first_col_dim
2629     }
2630     &
2631   }
2632   \omit

```

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

```

2633   \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`).

```

2634   \int_compare:nNnTF \l_@@_first_col_int = 0
2635   {
2636     \bool_if:NT \l_@@_code_before_bool
2637     {
2638       \hbox
2639       {
2640         \skip_horizontal:N -0.5\arrayrulewidth
2641         \pgfsys@markposition { \@@_env: - col - 1 }
2642         \skip_horizontal:N 0.5\arrayrulewidth
2643       }
2644     }
2645     \pgfpicture
2646     \pgfrememberpicturerepositiononpagetrue
2647     \pgfcoordinate { \@@_env: - col - 1 }
2648     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
2649     \str_if_empty:NF \l_@@_name_str
2650     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2651     \endpgfpicture
2652   }
2653   {
2654     \bool_if:NT \l_@@_code_before_bool
2655     {
2656       \hbox
2657       {
2658         \skip_horizontal:N 0.5\arrayrulewidth
2659         \pgfsys@markposition { \@@_env: - col - 1 }
2660         \skip_horizontal:N -0.5\arrayrulewidth
2661       }
2662     }
2663     \pgfpicture
2664     \pgfrememberpicturerepositiononpagetrue
2665     \pgfcoordinate { \@@_env: - col - 1 }
2666     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }

```

```

2667     \str_if_empty:NF \l_@@_name_str
2668     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2669     \endpgfpicture
2670 }

```

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.

```

2671     \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2672     \bool_if:NF \l_@@_auto_columns_width_bool
2673     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2674     {
2675         \bool_lazy_and:nnTF
2676         \l_@@_auto_columns_width_bool
2677         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2678         { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2679         { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2680         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2681     }
2682     \skip_horizontal:N \g_tmpa_skip
2683     \hbox
2684     {
2685         \bool_if:NT \l_@@_code_before_bool
2686         {
2687             \hbox
2688             {
2689                 \skip_horizontal:N -0.5\arrayrulewidth
2690                 \pgfsys@markposition { \@@_env: - col - 2 }
2691                 \skip_horizontal:N 0.5\arrayrulewidth
2692             }
2693         }
2694         \pgfpicture
2695         \pgfrememberpicturepositiononpagetrue
2696         \pgfcoordinate { \@@_env: - col - 2 }
2697         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2698         \str_if_empty:NF \l_@@_name_str
2699         { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2700         \endpgfpicture
2701     }

```

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.

```

2702     \int_gset:Nn \g_tmpa_int 1
2703     \bool_if:NTF \g_@@_last_col_found_bool
2704     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2705     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2706     {
2707         &
2708         \omit
2709         \int_gincr:N \g_tmpa_int

```

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

```

2710     \skip_horizontal:N \g_tmpa_skip
2711     \bool_if:NT \l_@@_code_before_bool
2712     {
2713         \hbox
2714         {
2715             \skip_horizontal:N -0.5\arrayrulewidth
2716             \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2717             \skip_horizontal:N 0.5\arrayrulewidth
2718         }

```

2719 }

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

```
2720 \pgfpicture
2721 \pgfrememberpicturepositiononpagetrue
2722 \pgfcoordinate { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
2723 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2724 \str_if_empty:NF \l_@@_name_str
2725 {
2726 \pgfnodealias
2727 { \l_@@_name_str - col - \l_@@_succ:n \g_tmpa_int }
2728 { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
2729 }
2730 \endpgfpicture
2731 }
```

```
2732 &
2733 \omit
2734 \int_gincr:N \g_tmpa_int
2735 \skip_horizontal:N \g_tmpa_skip
2736 \bool_lazy_all:nT
2737 {
2738 \l_@@_NiceArray_bool
2739 { \bool_not_p:n \l_@@_NiceTabular_bool }
2740 { \clist_if_empty_p:N \l_@@_vlines_clist }
2741 { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2742 { ! \l_@@_bar_at_end_of_pream_bool }
2743 }
2744 { \skip_horizontal:N -\col@sep }
2745 \bool_if:NT \l_@@_code_before_bool
2746 {
2747 \hbox
2748 {
2749 \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.

```
2750 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2751 { \skip_horizontal:N -\arraycolsep }
2752 \pgfsys@markposition { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
2753 \skip_horizontal:N 0.5\arrayrulewidth
2754 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2755 { \skip_horizontal:N \arraycolsep }
2756 }
2757 }
2758 \pgfpicture
2759 \pgfrememberpicturepositiononpagetrue
2760 \pgfcoordinate { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
2761 {
2762 \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
2763 {
2764 \pgfpoint
2765 { - 0.5 \arrayrulewidth - \arraycolsep }
2766 \c_zero_dim
2767 }
2768 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2769 }
2770 \str_if_empty:NF \l_@@_name_str
2771 {
2772 \pgfnodealias
2773 { \l_@@_name_str - col - \l_@@_succ:n \g_tmpa_int }
2774 { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
```

```

2775     }
2776     \endpgfpicture

2777     \bool_if:NT \g_@@_last_col_found_bool
2778     {
2779         \hbox_overlap_right:n
2780         {
2781             \skip_horizontal:N \g_@@_width_last_col_dim
2782             \bool_if:NT \l_@@_code_before_bool
2783             {
2784                 \pgfsys@markposition
2785                 { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2786             }
2787             \pgfpicture
2788             \pgfrememberpicturepositiononpagetrue
2789             \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2790             \pgfpointorigin
2791             \str_if_empty:NF \l_@@_name_str
2792             {
2793                 \pgfnodealias
2794                 { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2795                 { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2796             }
2797             \endpgfpicture
2798         }
2799     }
2800     \cr
2801 }

```

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

```

2802 \tl_const:Nn \c_@@_preamble_first_col_tl
2803 {
2804     >
2805     {

```

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).

```

2806         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
2807         \bool_gset_true:N \g_@@_after_col_zero_bool
2808         \@@_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.

```

2809         \hbox_set:Nw \l_@@_cell_box
2810         \@@_math_toggle_token:
2811         \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”.

```

2812         \bool_lazy_and:nnT
2813         { \int_compare_p:nNn \c@iRow > 0 }
2814         {
2815             \bool_lazy_or_p:nn
2816             { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2817             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2818         }
2819         {
2820             \l_@@_code_for_first_col_tl
2821             \xglobal \colorlet { nicematrix-first-col } { . }
2822         }
2823     }

```

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`.

```

2824   l
2825   <
2826   {
2827     \@@_math_toggle_token:
2828     \hbox_set_end:
2829     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2830     \@@_adjust_size_box:
2831     \@@_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.

```

2832     \dim_gset:Nn \g_@@_width_first_col_dim
2833     { \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.

```

2834     \hbox_overlap_left:n
2835     {
2836       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2837       \@@_node_for_cell:
2838       { \box_use_drop:N \l_@@_cell_box }
2839       \skip_horizontal:N \l_@@_left_delim_dim
2840       \skip_horizontal:N \l_@@_left_margin_dim
2841       \skip_horizontal:N \l_@@_extra_left_margin_dim
2842     }
2843     \bool_gset_false:N \g_@@_empty_cell_bool
2844     \skip_horizontal:N -2\col@sep
2845   }
2846 }

```

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

```

2847 \tl_const:Nn \c_@@_preamble_last_col_tl
2848 {
2849   >
2850   {

```

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).

```

2851     \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.

```

2852     \bool_gset_true:N \g_@@_last_col_found_bool
2853     \int_gincr:N \c@jCol
2854     \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.

```

2855     \hbox_set:Nw \l_@@_cell_box
2856     \@@_math_toggle_token:
2857     \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”.

```

2858     \int_compare:nNnT \c@iRow > 0
2859     {
2860       \bool_lazy_or:nnT
2861       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2862       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2863       {
2864         \l_@@_code_for_last_col_tl
2865         \xglobal \colorlet { nicematrix-last-col } { . }
2866       }
2867     }
2868   }
2869   l

```

```

2870 <
2871 {
2872   \@@_math_toggle_token:
2873   \hbox_set_end:
2874   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2875   \@@_adjust_size_box:
2876   \@@_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.

```

2877   \dim_gset:Nn \g_@@_width_last_col_dim
2878   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2879   \skip_horizontal:N -2\col@sep

```

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

```

2880   \hbox_overlap_right:n
2881   {
2882     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2883     {
2884       \skip_horizontal:N \l_@@_right_delim_dim
2885       \skip_horizontal:N \l_@@_right_margin_dim
2886       \skip_horizontal:N \l_@@_extra_right_margin_dim
2887       \@@_node_for_cell:
2888     }
2889   }
2890   \bool_gset_false:N \g_@@_empty_cell_bool
2891 }
2892 }

```

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.

```

2893 \NewDocumentEnvironment { NiceArray } { }
2894 {
2895   \bool_set_true:N \l_@@_NiceArray_bool
2896   \str_if_empty:NT \g_@@_name_env_str
2897   { \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).

```

2898   \NiceArrayWithDelims . .
2899 }
2900 { \endNiceArrayWithDelims }

```

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

```

2901 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2902 {
2903   \NewDocumentEnvironment { #1 NiceArray } { }
2904   {
2905     \str_if_empty:NT \g_@@_name_env_str
2906     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2907     \@@_test_if_math_mode:
2908     \NiceArrayWithDelims #2 #3
2909   }
2910   { \endNiceArrayWithDelims }
2911 }
2912 \@@_def_env:nnn p ( )
2913 \@@_def_env:nnn b [ ]
2914 \@@_def_env:nnn B \{ \}
2915 \@@_def_env:nnn v | |
2916 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2917 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2918 {
2919   \bool_set_true:N \l_@@_Matrix_bool
2920   \use:c { #1 NiceArray }
2921   {
2922     *
2923     {
2924       \int_compare:nNnTF \l_@@_last_col_int < 0
2925         \c@MaxMatrixCols
2926         { \@@_pred:n \l_@@_last_col_int }
2927     }
2928     { > \@@_cell_begin:w #2 < \@@_cell_end: }
2929   }
2930 }
2931 \clist_map_inline:nn { { } , p , b , B , v , V }
2932 {
2933   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2934   {
2935     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2936     \tl_set:Nn \l_@@_type_of_col_tl c
2937     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2938     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2939   }
2940   { \use:c { end #1 NiceArray } }
2941 }

```

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

```

2942 \cs_new_protected:Npn \@@_NotEmpty:
2943 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

2944 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2945 {

```

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.

```

2946   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
2947     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
2948   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2949   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2950   \bool_set_true:N \l_@@_NiceTabular_bool
2951   \NiceArray { #2 }
2952 }
2953 { \endNiceArray }

2954 \cs_set_protected:Npn \@@_newcolumnntype #1
2955 {
2956   \cs_if_free:cT { NC @ find @ #1 }
2957     { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
2958   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
2959   \peek_meaning:NTF [
2960     { \newcol@ #1 }
2961     { \newcol@ #1 [ 0 ] }
2962   }

```

```

2963 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
2964 {

```

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}).


```

2965 \bool_if:NT \c_@@_tabularx_loaded_bool
2966 { \newcolumnntype { X } { \@@_X } }
2967 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
2968 \dim_zero_new:N \l_@@_width_dim
2969 \dim_set:Nn \l_@@_width_dim { #1 }
2970 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2971 \bool_set_true:N \l_@@_NiceTabular_bool
2972 \NiceArray { #3 }
2973 }
2974 { \endNiceArray }

2975 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
2976 {
2977 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2978 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
2979 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2980 \bool_set_true:N \l_@@_NiceTabular_bool
2981 \NiceArray { #3 }
2982 }
2983 { \endNiceArray }

```

After the construction of the array

```

2984 \cs_new_protected:Npn \@@_after_array:
2985 {
2986 \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.

```

2987 \bool_if:NT \g_@@_last_col_found_bool
2988 { \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`.

```

2989 \bool_if:NT \l_@@_last_col_without_value_bool
2990 { \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.

```

2991 \bool_if:NT \l_@@_last_row_without_value_bool
2992 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

2993 \tl_gput_right:Nx \g_@@_aux_tl
2994 {
2995 \seq_gset_from_clist:Nn \exp_not:N \c_@@_size_seq
2996 {
2997 \int_use:N \l_@@_first_row_int ,
2998 \int_use:N \c_iRow ,
2999 \int_use:N \g_@@_row_total_int ,
3000 \int_use:N \l_@@_first_col_int ,
3001 \int_use:N \c_jCol ,
3002 \int_use:N \g_@@_col_total_int
3003 }
3004 }

```

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`).

```

3005 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3006 {

```

```

3007     \tl_gput_right:Nx \g_@@_aux_tl
3008     {
3009         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3010         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3011     }
3012 }
3013 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3014 {
3015     \tl_gput_right:Nx \g_@@_aux_tl
3016     {
3017         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3018         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3019         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3020         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3021     }
3022 }

```

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

```

3023 \@@_create_diag_nodes:

```

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

```

3024 \pgfpicture
3025 \int_step_inline:nn \c@iRow
3026 {
3027     \pgfnodealias
3028     { \@@_env: - ##1 - last }
3029     { \@@_env: - ##1 - \int_use:N \c@jCol }
3030 }
3031 \int_step_inline:nn \c@jCol
3032 {
3033     \pgfnodealias
3034     { \@@_env: - last - ##1 }
3035     { \@@_env: - \int_use:N \c@iRow - ##1 }
3036 }
3037 \str_if_empty:NF \l_@@_name_str
3038 {
3039     \int_step_inline:nn \c@iRow
3040     {
3041         \pgfnodealias
3042         { \l_@@_name_str - ##1 - last }
3043         { \@@_env: - ##1 - \int_use:N \c@jCol }
3044     }
3045     \int_step_inline:nn \c@jCol
3046     {
3047         \pgfnodealias
3048         { \l_@@_name_str - last - ##1 }
3049         { \@@_env: - \int_use:N \c@iRow - ##1 }
3050     }
3051 }
3052 \endpgfpicture

```

By default, the diagonal lines will be parallelized⁶⁹. 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.

```

3053 \bool_if:NT \l_@@_parallelize_diags_bool
3054 {
3055     \int_gzero_new:N \g_@@_ddots_int
3056     \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`

⁶⁹It's possible to use the option `parallelize-diags` to disable this parallelization.

diagonals parallel to the first one. Similarly Δ_x and Δ_y of the first Δ diagonal.

```

3057 \dim_gzero_new:N \g_@@_delta_x_one_dim
3058 \dim_gzero_new:N \g_@@_delta_y_one_dim
3059 \dim_gzero_new:N \g_@@_delta_x_two_dim
3060 \dim_gzero_new:N \g_@@_delta_y_two_dim
3061 }
3062 \int_zero_new:N \l_@@_initial_i_int
3063 \int_zero_new:N \l_@@_initial_j_int
3064 \int_zero_new:N \l_@@_final_i_int
3065 \int_zero_new:N \l_@@_final_j_int
3066 \bool_set_false:N \l_@@_initial_open_bool
3067 \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values Δ_{radius} and $\Delta_{\text{interdots}}$ (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.

```

3068 \bool_if:NT \l_@@_small_bool
3069 {
3070 \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
3071 \dim_set:Nn \l_@@_interdots_dim { 0.25 em }

```

The dimension $\Delta_{\text{xdots_shorten}}$ 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.

```

3072 \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
3073 }

```

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

```

3074 \@@_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 $\Delta_{\text{corners_cells_seq}}$ which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

3075 \@@_compute_corners:

```

The sequence $\Delta_{\text{pos_of_blocks_seq}}$ must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3076 \@@_adjust_pos_of_blocks_seq:
3077 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3078 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3079 \bool_if:NT \c_@@_tikz_loaded_bool
3080 {
3081 \tikzset
3082 {
3083 every~picture / .style =
3084 {
3085 overlay ,
3086 remember~picture ,
3087 name-prefix = \@@_env: -
3088 }
3089 }
3090 }
3091 \cs_set_eq:NN \ialign \@@_old_ialign:
3092 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3093 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3094 \cs_set_eq:NN \OverBrace \@@_OverBrace
3095 \cs_set_eq:NN \line \@@_line
3096 \g_@@_internal_code_after_tl
3097 \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.

```
3098 \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).

```
3099 \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:`.

```
3100 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3101 \scan_stop:
3102 \tl_gclear:N \g_nicematrix_code_after_tl
3103 \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.

```
3104 \tl_if_empty:N \g_nicematrix_code_before_tl
3105 {
```

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`.

```
3106 \cs_set_protected:Npn \rectanglecolor { }
3107 \cs_set_protected:Npn \columncolor { }
3108 \tl_gput_right:Nx \g_@@_aux_tl
3109 {
3110 \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3111 { \exp_not:N \g_nicematrix_code_before_tl }
3112 }
3113 \bool_set_true:N \l_@@_code_before_bool
3114 }

3115 \str_gclear:N \g_@@_name_env_str
3116 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁷⁰. 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`.

```
3117 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3118 }
```

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`.

```
3119 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3120 { \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).

⁷⁰e.g. `\color[rgb]{0.5,0.5,0}`

```

3121 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3122 {
3123   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3124   { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3125 }

```

The following command must *not* be protected.

```

3126 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3127 {
3128   { #1 }
3129   { #2 }
3130   {
3131     \int_compare:nNnTF { #3 } > { 99 }
3132     { \int_use:N \c@iRow }
3133     { #3 }
3134   }
3135   {
3136     \int_compare:nNnTF { #4 } > { 99 }
3137     { \int_use:N \c@jCol }
3138     { #4 }
3139   }
3140   { #5 }
3141 }

```

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).

```

3142 \AtBeginDocument
3143 {
3144   \cs_new_protected:Npx \@@_draw_dotted_lines:
3145   {
3146     \c_@@_pgfortikzpicture_tl
3147     \@@_draw_dotted_lines_i:
3148     \c_@@_endpgfortikzpicture_tl
3149   }
3150 }

```

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

```

3151 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3152 {
3153   \pgfrememberpicturerepositiononpagetrue
3154   \pgf@relevantforpicturesizefalse
3155   \g_@@_HVdotsfor_lines_tl
3156   \g_@@_Vdots_lines_tl
3157   \g_@@_Ddots_lines_tl
3158   \g_@@_Iddots_lines_tl
3159   \g_@@_Cdots_lines_tl
3160   \g_@@_Ldots_lines_tl
3161 }

3162 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3163 {
3164   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3165   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3166 }

```

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

```

3167 \pgfdeclareshape { @@_diag_node }
3168 {
3169   \savedanchor { \five }
3170   {

```

```

3171     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3172     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3173   }
3174   \anchor { 5 } { \five }
3175   \anchor { center } { \pgfpointorigin }
3176 }

```

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).

```

3177 \cs_new_protected:Npn \@@_create_diag_nodes:
3178 {
3179   \pgfpicture
3180   \pgfrememberpicturepositiononpagetrue
3181   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3182   {
3183     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3184     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3185     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3186     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3187     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3188     \dim_set_eq:NN \l_tmpc_dim \pgf@x
3189     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3190     \dim_set_eq:NN \l_tmpd_dim \pgf@y
3191     \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 `@â_diag_node`) that we will construct.

```

3192     \dim_set:Nn \l_tmpa_dim { ( \l_tmpc_dim - \l_tmpa_dim ) / 2 }
3193     \dim_set:Nn \l_tmpb_dim { ( \l_tmpd_dim - \l_tmpb_dim ) / 2 }
3194     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3195     \str_if_empty:NF \l_@@_name_str
3196     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3197   }

```

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

```

3198   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3199   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3200   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3201   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3202   \pgfcoordinate
3203   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3204   \pgfnodealias
3205   { \@@_env: - last }
3206   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3207   \str_if_empty:NF \l_@@_name_str
3208   {
3209     \pgfnodealias
3210     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3211     { \@@_env: - \int_use:N \l_tmpa_int }
3212     \pgfnodealias
3213     { \l_@@_name_str - last }
3214     { \@@_env: - last }
3215   }
3216   \endpgfpicture
3217 }

```

We draw the dotted lines

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

its right.

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

The command `\l_@@_find_extremities_of_line:nnnn` takes four arguments:

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

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
3218 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3219 {
```

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

```
3220 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3221 \int_set:Nn \l_@@_initial_i_int { #1 }
3222 \int_set:Nn \l_@@_initial_j_int { #2 }
3223 \int_set:Nn \l_@@_final_i_int { #1 }
3224 \int_set:Nn \l_@@_final_j_int { #2 }
```

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

```
3225 \bool_set_false:N \l_@@_stop_loop_bool
3226 \bool_do_until:Nn \l_@@_stop_loop_bool
3227 {
3228   \int_add:Nn \l_@@_final_i_int { #3 }
3229   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3230 \bool_set_false:N \l_@@_final_open_bool
3231 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3232 {
3233   \int_compare:nNnTF { #3 } = 1
3234   { \bool_set_true:N \l_@@_final_open_bool }
3235   {
3236     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3237     { \bool_set_true:N \l_@@_final_open_bool }
3238   }
3239 }
3240 {
3241   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3242   {
3243     \int_compare:nNnTF { #4 } = { -1 }
3244     { \bool_set_true:N \l_@@_final_open_bool }
3245   }
3246   {
```

```

3247         \int_compare:nNtT \l_@@_final_j_int > \l_@@_col_max_int
3248         {
3249             \int_compare:nNtT { #4 } = 1
3250             { \bool_set_true:N \l_@@_final_open_bool }
3251         }
3252     }
3253 }
3254 \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.

```

3255 {

```

We do a step backwards.

```

3256     \int_sub:Nn \l_@@_final_i_int { #3 }
3257     \int_sub:Nn \l_@@_final_j_int { #4 }
3258     \bool_set_true:N \l_@@_stop_loop_bool
3259 }

```

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`.

```

3260 {
3261     \cs_if_exist:cTF
3262     {
3263         @@ _ dotted _
3264         \int_use:N \l_@@_final_i_int -
3265         \int_use:N \l_@@_final_j_int
3266     }
3267     {
3268         \int_sub:Nn \l_@@_final_i_int { #3 }
3269         \int_sub:Nn \l_@@_final_j_int { #4 }
3270         \bool_set_true:N \l_@@_final_open_bool
3271         \bool_set_true:N \l_@@_stop_loop_bool
3272     }
3273     {
3274         \cs_if_exist:cTF
3275         {
3276             pgf @ sh @ ns @ \@@_env:
3277             - \int_use:N \l_@@_final_i_int
3278             - \int_use:N \l_@@_final_j_int
3279         }
3280         { \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.

```

3281     {
3282         \cs_set:cpn
3283         {
3284             @@ _ dotted _
3285             \int_use:N \l_@@_final_i_int -
3286             \int_use:N \l_@@_final_j_int
3287         }
3288         { }
3289     }
3290 }
3291 }
3292 }

```

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

```

3293     \bool_set_false:N \l_@@_stop_loop_bool

```



```

3294 \bool_do_until:Nn \l_@@_stop_loop_bool
3295 {
3296   \int_sub:Nn \l_@@_initial_i_int { #3 }
3297   \int_sub:Nn \l_@@_initial_j_int { #4 }
3298   \bool_set_false:N \l_@@_initial_open_bool
3299   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3300   {
3301     \int_compare:nNnTF { #3 } = 1
3302     { \bool_set_true:N \l_@@_initial_open_bool }
3303     {
3304       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3305       { \bool_set_true:N \l_@@_initial_open_bool }
3306     }
3307   }
3308   {
3309     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3310     {
3311       \int_compare:nNnT { #4 } = 1
3312       { \bool_set_true:N \l_@@_initial_open_bool }
3313     }
3314     {
3315       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3316       {
3317         \int_compare:nNnT { #4 } = { -1 }
3318         { \bool_set_true:N \l_@@_initial_open_bool }
3319       }
3320     }
3321   }
3322   \bool_if:NnTF \l_@@_initial_open_bool
3323   {
3324     \int_add:Nn \l_@@_initial_i_int { #3 }
3325     \int_add:Nn \l_@@_initial_j_int { #4 }
3326     \bool_set_true:N \l_@@_stop_loop_bool
3327   }
3328   {
3329     \cs_if_exist:cTF
3330     {
3331       @@ _ dotted _
3332       \int_use:N \l_@@_initial_i_int -
3333       \int_use:N \l_@@_initial_j_int
3334     }
3335     {
3336       \int_add:Nn \l_@@_initial_i_int { #3 }
3337       \int_add:Nn \l_@@_initial_j_int { #4 }
3338       \bool_set_true:N \l_@@_initial_open_bool
3339       \bool_set_true:N \l_@@_stop_loop_bool
3340     }
3341     {
3342       \cs_if_exist:cTF
3343       {
3344         pgf @ sh @ ns @ \@@_env:
3345         - \int_use:N \l_@@_initial_i_int
3346         - \int_use:N \l_@@_initial_j_int
3347       }
3348       { \bool_set_true:N \l_@@_stop_loop_bool }
3349       {
3350         \cs_set:cpn
3351         {
3352           @@ _ dotted _
3353           \int_use:N \l_@@_initial_i_int -
3354           \int_use:N \l_@@_initial_j_int
3355         }
3356         { }

```

```

3357         }
3358     }
3359 }
3360 }

```

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.

```

3361 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3362 {
3363     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

3364     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3365     { \int_use:N \l_@@_final_i_int }
3366     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3367     { } % for the name of the block
3368 }
3369 }

```

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).

```

3370 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3371 {
3372     \int_set:Nn \l_@@_row_min_int 1
3373     \int_set:Nn \l_@@_col_min_int 1
3374     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3375     \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`.

```

3376 \seq_map_inline:Nn \g_@@_submatrix_seq
3377 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
3378 }

```

#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.

```

3379 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3380 {
3381     \bool_if:nT
3382     {
3383         \int_compare_p:n { #3 <= #1 }
3384         && \int_compare_p:n { #1 <= #5 }
3385         && \int_compare_p:n { #4 <= #2 }
3386         && \int_compare_p:n { #2 <= #6 }
3387     }
3388     {
3389         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3390         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3391         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3392         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3393     }
3394 }

```

```

3395 \cs_new_protected:Npn \@@_set_initial_coords:
3396 {
3397     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3398     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3399 }
3400 \cs_new_protected:Npn \@@_set_final_coords:
3401 {

```

```

3402 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3403 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3404 }
3405 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3406 {
3407   \pgfpointanchor
3408   {
3409     \@@_env:
3410     - \int_use:N \l_@@_initial_i_int
3411     - \int_use:N \l_@@_initial_j_int
3412   }
3413   { #1 }
3414   \@@_set_initial_coords:
3415 }
3416 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3417 {
3418   \pgfpointanchor
3419   {
3420     \@@_env:
3421     - \int_use:N \l_@@_final_i_int
3422     - \int_use:N \l_@@_final_j_int
3423   }
3424   { #1 }
3425   \@@_set_final_coords:
3426 }
3427 \cs_new_protected:Npn \@@_open_x_initial_dim:
3428 {
3429   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3430   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3431   {
3432     \cs_if_exist:cT
3433     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3434     {
3435       \pgfpointanchor
3436       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3437       { west }
3438       \dim_set:Nn \l_@@_x_initial_dim
3439       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3440     }
3441   }

```

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

```

3442 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3443 {
3444   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3445   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3446   \dim_add:Nn \l_@@_x_initial_dim \col@sep
3447 }
3448 }
3449 \cs_new_protected:Npn \@@_open_x_final_dim:
3450 {
3451   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3452   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3453   {
3454     \cs_if_exist:cT
3455     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3456     {
3457       \pgfpointanchor
3458       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3459       { east }
3460       \dim_set:Nn \l_@@_x_final_dim
3461       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3462     }

```

```
3463 }
```

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

```
3464 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3465 {
3466   \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3467   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3468   \dim_sub:Nn \l_@@_x_final_dim \col@sep
3469 }
3470 }
```

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.

```
3471 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3472 {
3473   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3474   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3475   {
3476     \@@_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.

```
3477   \group_begin:
3478   \int_compare:nNnTF { #1 } = 0
3479   { \color { nicematrix-first-row } }
3480   {
```

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.

```
3481     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3482     { \color { nicematrix-last-row } }
3483   }
3484   \keys_set:nn { NiceMatrix / xdots } { #3 }
3485   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3486   \@@_actually_draw_Ldots:
3487   \group_end:
3488 }
3489 }
```

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`.

```
3490 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3491 {
3492   \bool_if:NTF \l_@@_initial_open_bool
3493   {
3494     \@@_open_x_initial_dim:
3495     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3496     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3497   }
3498   { \@@_set_initial_coords_from_anchor:n { base-east } }
3499   \bool_if:NTF \l_@@_final_open_bool
```

```

3500 {
3501   \@@_open_x_final_dim:
3502   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3503   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3504 }
3505 { \@@_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 (?).

```

3506   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3507   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3508   \@@_draw_line:
3509 }

```

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.

```

3510 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3511 {
3512   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3513   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3514   {
3515     \@@_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.

```

3516     \group_begin:
3517     \int_compare:nNnTF { #1 } = 0
3518     { \color { nicematrix-first-row } }
3519     {

```

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.

```

3520       \int_compare:nNnT { #1 } = \l_@@_last_row_int
3521       { \color { nicematrix-last-row } }
3522     }
3523     \keys_set:nn { NiceMatrix / xdots } { #3 }
3524     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3525     \@@_actually_draw_Cdots:
3526   \group_end:
3527 }
3528 }

```

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`.

```

3529 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3530 {
3531   \bool_if:NTF \l_@@_initial_open_bool
3532   { \@@_open_x_initial_dim: }
3533   { \@@_set_initial_coords_from_anchor:n { mid~east } }
3534   \bool_if:NTF \l_@@_final_open_bool
3535   { \@@_open_x_final_dim: }
3536   { \@@_set_final_coords_from_anchor:n { mid~west } }
3537   \bool_lazy_and:nnTF

```

```

3538 \l_@@_initial_open_bool
3539 \l_@@_final_open_bool
3540 {
3541   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3542   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3543   \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
3544   \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3545   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3546 }
3547 {
3548   \bool_if:NT \l_@@_initial_open_bool
3549   { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3550   \bool_if:NT \l_@@_final_open_bool
3551   { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3552 }
3553 \@@_draw_line:
3554 }
3555 \cs_new_protected:Npn \@@_open_y_initial_dim:
3556 {
3557   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3558   \dim_set:Nn \l_@@_y_initial_dim
3559   { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3560   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3561   {
3562     \cs_if_exist:cT
3563     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3564     {
3565       \pgfpointanchor
3566       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3567       { north }
3568       \dim_set:Nn \l_@@_y_initial_dim
3569       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3570     }
3571   }
3572 }
3573 \cs_new_protected:Npn \@@_open_y_final_dim:
3574 {
3575   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3576   \dim_set:Nn \l_@@_y_final_dim
3577   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3578   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3579   {
3580     \cs_if_exist:cT
3581     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3582     {
3583       \pgfpointanchor
3584       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3585       { south }
3586       \dim_set:Nn \l_@@_y_final_dim
3587       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3588     }
3589   }
3590 }

```

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.

```

3591 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3592 {
3593   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3594   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3595   {
3596     \@@_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.

```

3597     \group_begin:
3598     \int_compare:nNnTF { #2 } = 0
3599     { \color { nicematrix-first-col } }
3600     {
3601         \int_compare:nNnT { #2 } = \l_@@_last_col_int
3602         { \color { nicematrix-last-col } }
3603     }
3604     \keys_set:nn { NiceMatrix / xdots } { #3 }
3605     \tl_if_empty:VF \l_@@_xdots_color_tl
3606     { \color { \l_@@_xdots_color_tl } }
3607     \@@_actually_draw_Vdots:
3608 \group_end:
3609 }
3610 }
```

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`.

```

3611 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3612 {
```

The boolean `\l_tmpa_bool` indicates whether the column is of type 1 or may be considered as if.

```

3613     \bool_set_false:N \l_tmpa_bool
```

First the case when the line is closed on both ends.

```

3614     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3615     {
3616         \@@_set_initial_coords_from_anchor:n { south-west }
3617         \@@_set_final_coords_from_anchor:n { north-west }
3618         \bool_set:Nn \l_tmpa_bool
3619         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3620     }
```

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

```

3621     \bool_if:NTF \l_@@_initial_open_bool
3622     \@@_open_y_initial_dim:
3623     { \@@_set_initial_coords_from_anchor:n { south } }
3624     \bool_if:NTF \l_@@_final_open_bool
3625     \@@_open_y_final_dim:
3626     { \@@_set_final_coords_from_anchor:n { north } }
3627     \bool_if:NTF \l_@@_initial_open_bool
3628     {
3629         \bool_if:NTF \l_@@_final_open_bool
3630         {
3631             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3632             \dim_set_eq:NN \l_tmpa_dim \pgf@x
3633             \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3634             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3635             \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.

```

3636         \int_compare:nNnT \l_@@_last_col_int > { -2 }
3637         {
3638             \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3639             {
3640                 \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3641                 \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3642                 \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3643                 \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3644             }
3645         }
3646     }
3647     { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3648 }
3649 {
3650     \bool_if:NTF \l_@@_final_open_bool
3651     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3652 }

```

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.

```

3653         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3654         {
3655             \dim_set:Nn \l_@@_x_initial_dim
3656             {
3657                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3658                 \l_@@_x_initial_dim \l_@@_x_final_dim
3659             }
3660             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3661         }
3662     }
3663 }
3664 \@@_draw_line:
3665 }

```

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.

```

3666 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3667 {
3668     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3669     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3670     {
3671         \@@_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.

```

3672     \group_begin:
3673     \keys_set:nn { NiceMatrix / xdots } { #3 }
3674     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3675     \@@_actually_draw_Ddots:
3676     \group_end:
3677 }
3678 }

```

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.

```

3679 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3680 {
3681   \bool_if:NTF \l_@@_initial_open_bool
3682   {
3683     \@@_open_y_initial_dim:
3684     \@@_open_x_initial_dim:
3685   }
3686   { \@@_set_initial_coords_from_anchor:n { south-east } }
3687   \bool_if:NTF \l_@@_final_open_bool
3688   {
3689     \@@_open_x_final_dim:
3690     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3691   }
3692   { \@@_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.

```

3693   \bool_if:NT \l_@@_parallelize_diags_bool
3694   {
3695     \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).

```

3696     \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.

```

3697     {
3698       \dim_gset:Nn \g_@@_delta_x_one_dim
3699       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3700       \dim_gset:Nn \g_@@_delta_y_one_dim
3701       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3702     }

```

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.

```

3703     {
3704       \dim_set:Nn \l_@@_y_final_dim
3705       {
3706         \l_@@_y_initial_dim +
3707         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3708         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3709       }
3710     }
3711   }
3712   \@@_draw_line:
3713 }

```

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.

```

3714 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3715 {
3716   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3717   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }

```

```

3718 {
3719   \@@_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.

```

3720   \group_begin:
3721     \keys_set:nn { NiceMatrix / xdots } { #3 }
3722     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3723     \@@_actually_draw_Iddots:
3724   \group_end:
3725 }
3726 }

```

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.`

```

3727 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3728 {
3729   \bool_if:NTF \l_@@_initial_open_bool
3730   {
3731     \@@_open_y_initial_dim:
3732     \@@_open_x_initial_dim:
3733   }
3734   { \@@_set_initial_coords_from_anchor:n { south-west } }
3735   \bool_if:NTF \l_@@_final_open_bool
3736   {
3737     \@@_open_y_final_dim:
3738     \@@_open_x_final_dim:
3739   }
3740   { \@@_set_final_coords_from_anchor:n { north-east } }
3741   \bool_if:NT \l_@@_parallelize_diags_bool
3742   {
3743     \int_gincr:N \g_@@_iddots_int
3744     \int_compare:nNnTF \g_@@_iddots_int = 1
3745     {
3746       \dim_gset:Nn \g_@@_delta_x_two_dim
3747       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3748       \dim_gset:Nn \g_@@_delta_y_two_dim
3749       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3750     }
3751     {
3752       \dim_set:Nn \l_@@_y_final_dim
3753       {
3754         \l_@@_y_initial_dim +
3755         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3756         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3757       }
3758     }
3759   }
3760   \@@_draw_line:
3761 }

```

The actual instructions for drawing the dotted lines with Tikz

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

3762 \cs_new_protected:Npn \@@_draw_line:
3763 {
3764   \pgfrememberpicturepositiononpagetrue
3765   \pgf@relevantforpicturesizefalse
3766   \bool_lazy_or:nnTF
3767   { \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.

```

3768   \l_@@_dotted_bool
3769   \@@_draw_standard_dotted_line:
3770   \@@_draw_unstandard_dotted_line:
3771 }

```

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.

```

3772 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
3773 {
3774   \begin { scope }
3775   \exp_args:No \@@_draw_unstandard_dotted_line:n
3776   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3777 }

```

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.

```

3778 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
3779 {
3780   \@@_draw_unstandard_dotted_line:nVV
3781   { #1 }
3782   \l_@@_xdots_up_tl
3783   \l_@@_xdots_down_tl
3784 }
3785 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
3786 {
3787   \draw
3788   [
3789     #1 ,
3790     shorten~> = \l_@@_xdots_shorten_dim ,
3791     shorten~< = \l_@@_xdots_shorten_dim ,
3792   ]
3793   ( \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).

```

3794   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3795   node [ sloped , below ] { $ \scriptstyle #3 $ }
3796   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3797 \end { scope }
3798 }
3799 \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).

```

3800 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3801 {
3802   \bool_lazy_and:nnF
3803     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3804     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3805   {
3806     \pgfscope
3807     \pgftransformshift
3808       {
3809         \pgfpointlineattime { 0.5 }
3810         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3811         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3812       }
3813     \pgftransformrotate
3814       {
3815         \fp_eval:n
3816           {
3817             atand
3818             (
3819               \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3820               \l_@@_x_final_dim - \l_@@_x_initial_dim
3821             )
3822           }
3823       }
3824     \pgfnode
3825       { rectangle }
3826       { south }
3827       {
3828         \c_math_toggle_token
3829         \scriptstyle \l_@@_xdots_up_tl
3830         \c_math_toggle_token
3831       }
3832       { }
3833       { \pgfusepath { } }
3834     \pgfnode
3835       { rectangle }
3836       { north }
3837       {
3838         \c_math_toggle_token
3839         \scriptstyle \l_@@_xdots_down_tl
3840         \c_math_toggle_token
3841       }
3842       { }
3843       { \pgfusepath { } }
3844     \endpgfscope
3845   }
3846   \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.

```

3847   \dim_zero_new:N \l_@@_l_dim
3848   \dim_set:Nn \l_@@_l_dim
3849     {
3850       \fp_to_dim:n
3851         {
3852           sqrt
3853             (
3854               ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3855               +
3856               ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3857             )
3858         }

```

```
3859     }
```

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.

```
3860     \bool_lazy_or:nnF
3861     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3862     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3863     \@@_draw_standard_dotted_line_i:
3864   \group_end:
3865 }
3866 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3867 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3868 {
```

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

```
3869   \bool_if:NTF \l_@@_initial_open_bool
3870   {
3871     \bool_if:NTF \l_@@_final_open_bool
3872     {
3873       \int_set:Nn \l_tmpa_int
3874       { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3875     }
3876     {
3877       \int_set:Nn \l_tmpa_int
3878       {
3879         \dim_ratio:nn
3880         { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3881         \l_@@_inter_dots_dim
3882       }
3883     }
3884   }
3885   {
3886     \bool_if:NTF \l_@@_final_open_bool
3887     {
3888       \int_set:Nn \l_tmpa_int
3889       {
3890         \dim_ratio:nn
3891         { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3892         \l_@@_inter_dots_dim
3893       }
3894     }
3895     {
3896       \int_set:Nn \l_tmpa_int
3897       {
3898         \dim_ratio:nn
3899         { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3900         \l_@@_inter_dots_dim
3901       }
3902     }
3903   }
```

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

```
3904   \dim_set:Nn \l_tmpa_dim
3905   {
3906     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3907     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3908   }
3909   \dim_set:Nn \l_tmpb_dim
3910   {
3911     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
```

```

3912     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3913 }

```

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`.

```

3914 \int_set:Nn \l_tmpb_int
3915 {
3916   \bool_if:NTF \l_@@_initial_open_bool
3917     { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3918     { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3919 }

```

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.

```

3920 \dim_gadd:Nn \l_@@_x_initial_dim
3921 {
3922   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3923   \dim_ratio:nn
3924   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3925   { 2 \l_@@_l_dim }
3926   * \l_tmpb_int
3927 }
3928 \dim_gadd:Nn \l_@@_y_initial_dim
3929 {
3930   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3931   \dim_ratio:nn
3932   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3933   { 2 \l_@@_l_dim }
3934   * \l_tmpb_int
3935 }
3936 \pgf@relevantforpicturesizefalse
3937 \int_step_inline:nnn 0 \l_tmpa_int
3938 {
3939   \pgfpathcircle
3940   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3941   { \l_@@_radius_dim }
3942   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3943   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3944 }
3945 \pgfusepathqfill
3946 }

```

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 `\AtBeginDocument` and the *arg spec* will be rescanned.

```

3947 \AtBeginDocument
3948 {
3949   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3950   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

3951 \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3952 {
3953   \int_compare:nNnTF \c@jCol = 0
3954   { \@@_error:nn { in~first~col } \Ldots }
3955   {
3956     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3957     { \@@_error:nn { in~last~col } \Ldots }
3958     {
3959       \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
3960       { #1 , down = #2 , up = #3 }
3961     }
3962   }
3963   \bool_if:NF \l_@@_nullify_dots_bool
3964   { \phantom { \ensuremath { \@@_old_ldots } } }
3965   \bool_gset_true:N \g_@@_empty_cell_bool
3966 }

3967 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3968 {
3969   \int_compare:nNnTF \c@jCol = 0
3970   { \@@_error:nn { in~first~col } \Cdots }
3971   {
3972     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3973     { \@@_error:nn { in~last~col } \Cdots }
3974     {
3975       \@@_instruction_of_type:nnn \c_false_bool { \Cdots }
3976       { #1 , down = #2 , up = #3 }
3977     }
3978   }
3979   \bool_if:NF \l_@@_nullify_dots_bool
3980   { \phantom { \ensuremath { \@@_old_cdots } } }
3981   \bool_gset_true:N \g_@@_empty_cell_bool
3982 }

3983 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
3984 {
3985   \int_compare:nNnTF \c@iRow = 0
3986   { \@@_error:nn { in~first~row } \Vdots }
3987   {
3988     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
3989     { \@@_error:nn { in~last~row } \Vdots }
3990     {
3991       \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
3992       { #1 , down = #2 , up = #3 }
3993     }
3994   }
3995   \bool_if:NF \l_@@_nullify_dots_bool
3996   { \phantom { \ensuremath { \@@_old_vdots } } }
3997   \bool_gset_true:N \g_@@_empty_cell_bool
3998 }

3999 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4000 {
4001   \int_case:nnF \c@iRow
4002   {
4003     0 { \@@_error:nn { in~first~row } \Ddots }
4004     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4005   }
4006   {
4007     \int_case:nnF \c@jCol
4008     {

```

```

4009         0 { \@@_error:nn { in~first~col } \Ddots }
4010     \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4011 }
4012 {
4013     \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4014     \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4015     { #1 , down = #2 , up = #3 }
4016 }
4017
4018 }
4019 \bool_if:NF \l_@@_nullify_dots_bool
4020 { \phantom { \ensuremath { \@@_old_ddots } } }
4021 \bool_gset_true:N \g_@@_empty_cell_bool
4022 }

4023 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4024 {
4025     \int_case:nnF \c@iRow
4026     {
4027         0 { \@@_error:nn { in~first~row } \Iddots }
4028         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4029     }
4030     {
4031         \int_case:nnF \c@jCol
4032         {
4033             0 { \@@_error:nn { in~first~col } \Iddots }
4034             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4035         }
4036         {
4037             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4038             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4039             { #1 , down = #2 , up = #3 }
4040         }
4041     }
4042     \bool_if:NF \l_@@_nullify_dots_bool
4043     { \phantom { \ensuremath { \@@_old_iddots } } }
4044     \bool_gset_true:N \g_@@_empty_cell_bool
4045 }
4046 }

```

End of the \AtBeginDocument.

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

```

4047 \keys_define:nn { NiceMatrix / Ddots }
4048 {
4049     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4050     draw-first .default:n = true ,
4051     draw-first .value_forbidden:n = true
4052 }

```

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

```

4053 \cs_new_protected:Npn \@@_Hspace:
4054 {
4055     \bool_gset_true:N \g_@@_empty_cell_bool
4056     \hspace
4057 }

```

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.

```

4058 \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`.

```

4059 \cs_new:Npn \@@Hdotsfor:
4060 {
4061   \bool_lazy_and:nnTF
4062     { \int_compare_p:nNn \c@jCol = 0 }
4063     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4064     {
4065       \bool_if:NTF \g_@@_after_col_zero_bool
4066       {
4067         \multicolumn { 1 } { c } { }
4068         \@@Hdotsfor_i
4069       }
4070       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4071     }
4072     {
4073       \multicolumn { 1 } { c } { }
4074       \@@Hdotsfor_i
4075     }
4076 }

```

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:`).

```

4077 \AtBeginDocument
4078 {
4079   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4080   \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.

```

4081   \exp_args:NNV \NewDocumentCommand \@@Hdotsfor_i \l_@@_argspec_tl
4082   {
4083     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4084     {
4085       \@@Hdotsfor:nnnn
4086       { \int_use:N \c@iRow }
4087       { \int_use:N \c@jCol }
4088       { #2 }
4089       {
4090         #1 , #3 ,
4091         down = \exp_not:n { #4 } ,
4092         up = \exp_not:n { #5 }
4093       }
4094     }
4095     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4096   }
4097 }

```

Enf of `\AtBeginDocument`.

```

4098 \cs_new_protected:Npn \@@Hdotsfor:nnnn #1 #2 #3 #4
4099 {
4100   \bool_set_false:N \l_@@_initial_open_bool
4101   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4102   \int_set:Nn \l_@@_initial_i_int { #1 }
4103   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4104   \int_compare:nNnTF { #2 } = 1
4105   {

```

```

4106     \int_set:Nn \l_@@_initial_j_int 1
4107     \bool_set_true:N \l_@@_initial_open_bool
4108   }
4109   {
4110     \cs_if_exist:cTF
4111     {
4112       pgf @ sh @ ns @ \@@_env:
4113       - \int_use:N \l_@@_initial_i_int
4114       - \int_eval:n { #2 - 1 }
4115     }
4116     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4117     {
4118       \int_set:Nn \l_@@_initial_j_int { #2 }
4119       \bool_set_true:N \l_@@_initial_open_bool
4120     }
4121   }
4122   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4123   {
4124     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4125     \bool_set_true:N \l_@@_final_open_bool
4126   }
4127   {
4128     \cs_if_exist:cTF
4129     {
4130       pgf @ sh @ ns @ \@@_env:
4131       - \int_use:N \l_@@_final_i_int
4132       - \int_eval:n { #2 + #3 }
4133     }
4134     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4135     {
4136       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4137       \bool_set_true:N \l_@@_final_open_bool
4138     }
4139   }
4140   \group_begin:
4141   \int_compare:nNnTF { #1 } = 0
4142   { \color { nicematrix-first-row } }
4143   {
4144     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4145     { \color { nicematrix-last-row } }
4146   }
4147   \keys_set:nn { NiceMatrix / xdots } { #4 }
4148   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4149   \@@_actually_draw_Ldots:
4150   \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).

```

4151     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4152     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4153   }

4154   \AtBeginDocument
4155   {
4156     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4157     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4158     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4159     {
4160       \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4161       {
4162         \@@_Vdotsfor:nnnn

```

```

4163         { \int_use:N \c@iRow }
4164         { \int_use:N \c@jCol }
4165         { #2 }
4166         {
4167             #1 , #3 ,
4168             down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4169         }
4170     }
4171 }
4172 }

```

Enf of \AtBeginDocument.

```

4173 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4174 {
4175     \bool_set_false:N \l_@@_initial_open_bool
4176     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

4177     \int_set:Nn \l_@@_initial_j_int { #2 }
4178     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

4179     \int_compare:nNnTF #1 = 1
4180     {
4181         \int_set:Nn \l_@@_initial_i_int 1
4182         \bool_set_true:N \l_@@_initial_open_bool
4183     }
4184     {
4185         \cs_if_exist:cTF
4186         {
4187             pgf @ sh @ ns @ \@@_env:
4188             - \int_eval:n { #1 - 1 }
4189             - \int_use:N \l_@@_initial_j_int
4190         }
4191         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4192         {
4193             \int_set:Nn \l_@@_initial_i_int { #1 }
4194             \bool_set_true:N \l_@@_initial_open_bool
4195         }
4196     }
4197     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4198     {
4199         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4200         \bool_set_true:N \l_@@_final_open_bool
4201     }
4202     {
4203         \cs_if_exist:cTF
4204         {
4205             pgf @ sh @ ns @ \@@_env:
4206             - \int_eval:n { #1 + #3 }
4207             - \int_use:N \l_@@_final_j_int
4208         }
4209         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4210         {
4211             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4212             \bool_set_true:N \l_@@_final_open_bool
4213         }
4214     }
4215     \group_begin:
4216     \int_compare:nNnTF { #2 } = 0
4217     { \color { nicematrix-first-col } }
4218     {
4219         \int_compare:nNnT { #2 } = \g_@@_col_total_int
4220         { \color { nicematrix-last-col } }

```

```

4221     }
4222     \keys_set:nn { NiceMatrix / xdots } { #4 }
4223     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4224     \@@_actually_draw_Vdots:
4225     \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).

```

4226     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4227     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4228 }

```

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

```

4229 \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).⁷¹

```

4230 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4231 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

4232 \AtBeginDocument
4233 {
4234     \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4235     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4236     \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4237     {
4238         \group_begin:
4239         \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4240         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4241         \use:e
4242         {
4243             \@@_line_i:nn
4244             { \@@_double_int_eval:n #2 \q_stop }
4245             { \@@_double_int_eval:n #3 \q_stop }
4246         }
4247         \group_end:
4248     }
4249 }

4250 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4251 {
4252     \bool_set_false:N \l_@@_initial_open_bool
4253     \bool_set_false:N \l_@@_final_open_bool
4254     \bool_if:nTF
4255     {

```

⁷¹Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

4256     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4257     ||
4258     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4259   }
4260   {
4261     \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4262   }
4263   { \@@_draw_line_ii:nn { #1 } { #2 } }
4264 }
4265 \AtBeginDocument
4266 {
4267   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4268   {

```

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:`.

```

4269     \c_@@_pgfortikzpicture_tl
4270     \@@_draw_line_iii:nn { #1 } { #2 }
4271     \c_@@_endpgfortikzpicture_tl
4272   }
4273 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

4274 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4275 {
4276   \pgfrememberpicturepositiononpagetrue
4277   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4278   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4279   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4280   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4281   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4282   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4283   \@@_draw_line:
4284 }

```

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`

```

4285 \keys_define:nn { NiceMatrix / RowStyle }
4286 {
4287   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4288   cell-space-top-limit .initial:n = \c_zero_dim ,
4289   cell-space-top-limit .value_required:n = true ,
4290   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4291   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4292   cell-space-bottom-limit .value_required:n = true ,
4293   cell-space-limits .meta:n =
4294   {
4295     cell-space-top-limit = #1 ,
4296     cell-space-bottom-limit = #1 ,
4297   } ,
4298   color .tl_set:N = \l_tmpa_tl ,
4299   color .value_required:n = true ,
4300   bold .bool_set:N = \l_tmpa_bool ,
4301   bold .default:n = true ,
4302   bold .initial:n = false ,
4303   nb-rows .int_set:N = \l_@@_key_nb_rows_int ,
4304   nb-rows .value_required:n = true ,

```

```

4305     nb-rows .initial:n = 1 ,
4306     rowcolor .tl_set:N = \l_tmpc_tl ,
4307     rowcolor .value_required:n = true ,
4308     rowcolor .initial:n = \c_empty_tl ,
4309     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
4310 }

```

```

4311 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4312 {
4313     \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

4314     \tl_if_empty:NF \l_tmpc_tl
4315     {

```

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

```

4316         \tl_gput_right:Nx \g_nicematrix_code_before_tl
4317         {
4318             \@@_rectanglecolor
4319             { \l_tmpc_tl }
4320             { \int_use:N \c@iRow - \int_use:N \c@jCol }
4321             { \int_use:N \c@iRow - * }
4322         }

```

Then, the other rows (if there is several rows).

```

4323         \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4324         {
4325             \tl_gput_right:Nx \g_nicematrix_code_before_tl
4326             {
4327                 \@@_rowcolor
4328                 { \l_tmpc_tl }
4329                 {
4330                     \int_eval:n { \c@iRow + 1 }
4331                     - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4332                 }
4333             }
4334         }
4335     }
4336     \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4337     \tl_gput_right:Nx \g_@@_row_style_tl
4338     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4339     \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

4340     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4341     {
4342         \tl_gput_right:Nx \g_@@_row_style_tl
4343         {
4344             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4345             {
4346                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
4347                 { \dim_use:N \l_tmpa_dim }
4348             }
4349         }
4350     }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

4351     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4352     {
4353         \tl_gput_right:Nx \g_@@_row_style_tl
4354         {
4355             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4356             {
4357                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4358                 { \dim_use:N \l_tmpb_dim }
4359             }

```

```

4360     }
4361 }
\l_tmpa_tl is the value of the key color of \RowStyle.
4362 \tl_if_empty:NF \l_tmpa_tl
4363 {
4364     \tl_gput_right:Nx \g_@@_row_style_tl
4365     { \mode_leave_vertical: \exp_not:N \color { \l_tmpa_tl } }
4366 }
\l_tmpa_bool is the value of the key bold.
4367 \bool_if:NT \l_tmpa_bool
4368 {
4369     \tl_gput_right:Nn \g_@@_row_style_tl
4370     {
4371         \if_mode_math:
4372             \c_math_toggle_token
4373             \bfseries \boldmath
4374             \c_math_toggle_token
4375         \else:
4376             \bfseries \boldmath
4377         \fi:
4378     }
4379 }
4380 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4381 \g_@@_row_style_tl
4382 \ignorespaces
4383 }

```

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).

```

4384 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4385 {

```

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.

```

4386     \int_zero:N \l_tmpa_int
4387     \seq_map_indexed_inline:Nn \g_@@_colors_seq
4388     { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4389     \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

4390 {
4391   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4392   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4393 }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

4394 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4395 }

4396 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4397 \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`).

```

4398 \cs_new_protected:Npn \@@_actually_color:
4399 {
4400   \pgfpicture
4401   \pgf@relevantforpicturesizefalse
4402   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4403   {
4404     \color ##2
4405     \use:c { g_@@_color _ ##1 _tl }
4406     \tl_gclear:c { g_@@_color _ ##1 _tl }
4407     \pgfusepath { fill }
4408   }
4409   \endpgfpicture
4410 }

4411 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4412 {
4413   \tl_set:Nn \l_@@_rows_tl { #1 }
4414   \tl_set:Nn \l_@@_cols_tl { #2 }
4415   \@@_cartesian_path:
4416 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

4417 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4418 {
4419   \tl_if_blank:nF { #2 }
4420   {
4421     \@@_add_to_colors_seq:xn
4422     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4423     { \@@_cartesian_color:nn { #3 } { - } }
4424   }
4425 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

4426 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4427 {
4428   \tl_if_blank:nF { #2 }
4429   {
4430     \@@_add_to_colors_seq:xn
4431     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4432     { \@@_cartesian_color:nn { - } { #3 } }
4433   }
4434 }

```


Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

4435 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4436 {
4437   \tl_if_blank:nF { #2 }
4438   {
4439     \@@_add_to_colors_seq:xn
4440     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4441     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4442   }
4443 }

```

The last argument is the radius of the corners of the rectangle.

```

4444 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4445 {
4446   \tl_if_blank:nF { #2 }
4447   {
4448     \@@_add_to_colors_seq:xn
4449     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4450     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4451   }
4452 }

```

The last argument is the radius of the corners of the rectangle.

```

4453 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4454 {
4455   \@@_cut_on_hyphen:w #1 \q_stop
4456   \tl_clear_new:N \l_tmpc_tl
4457   \tl_clear_new:N \l_tmpd_tl
4458   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
4459   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
4460   \@@_cut_on_hyphen:w #2 \q_stop
4461   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
4462   \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`.

```

4463   \@@_cartesian_path:n { #3 }
4464 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

4465 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4466 {
4467   \clist_map_inline:nn { #3 }
4468   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4469 }

```

```

4470 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4471 {
4472   \int_step_inline:nn { \int_use:N \c@iRow }
4473   {
4474     \int_step_inline:nn { \int_use:N \c@jCol }
4475     {
4476       \int_if_even:nTF { ####1 + ##1 }
4477       { \@@_cellcolor [ #1 ] { #2 } }
4478       { \@@_cellcolor [ #1 ] { #3 } }
4479       { ##1 - ####1 }
4480     }
4481   }
4482 }

```

```

4483 \keys_define:nn { NiceMatrix / arraycolor }
4484 { except-corners .code:n = \@@_error:n { key~except-corners } }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns). The third argument is a optional argument which a list of pairs key-value.

```

4485 \NewDocumentCommand \@@_arraycolor { 0 { } m 0 { } }
4486 {
4487   \keys_set:nn { NiceMatrix / arraycolor } { #3 }
4488   \@@_rectanglecolor [ #1 ] { #2 }
4489   { 1 - 1 }
4490   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4491 }

```

```

4492 \keys_define:nn { NiceMatrix / rowcolors }
4493 {
4494   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4495   respect-blocks .default:n = true ,
4496   cols .tl_set:N = \l_@@_cols_tl ,
4497   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4498   restart .default:n = true ,
4499   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4500 }

```

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.

```

4501 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
4502 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

4503   \group_begin:
4504   \seq_clear_new:N \l_@@_colors_seq
4505   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
4506   \tl_clear_new:N \l_@@_cols_tl
4507   \tl_set:Nn \l_@@_cols_tl { - }
4508   \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).

```

4509   \int_zero_new:N \l_@@_color_int
4510   \int_set:Nn \l_@@_color_int 1
4511   \bool_if:NT \l_@@_respect_blocks_bool
4512   {

```

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`).

```

4513     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4514     \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
4515     { \@@_not_in_exterior_p:nnnnn ##1 }
4516   }
4517   \pgfpicture
4518   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

4519   \clist_map_inline:nn { #2 }
4520   {
4521     \tl_set:Nn \l_tmpa_tl { ##1 }

```

```

4522 \tl_if_in:NnTF \l_tmpa_tl { - }
4523 { \@@_cut_on_hyphen:w ##1 \q_stop }
4524 { \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.

```

4525 \int_set:Nn \l_tmpa_int \l_tmpa_tl
4526 \bool_if:NTF \l_@@_rowcolors_restart_bool
4527 { \int_set:Nn \l_@@_color_int 1 }
4528 { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
4529 \int_zero_new:N \l_tmpc_int
4530 \int_set:Nn \l_tmpc_int \l_tmpb_tl
4531 \int_do_until:nNnn \l_tmpa_int > \l_tmpc_int
4532 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4533 \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).

```

4534 \bool_if:NT \l_@@_respect_blocks_bool
4535 {
4536 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
4537 { \@@_intersect_our_row_p:nnnnn #####1 }
4538 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4539 }
4540 \tl_set:Nx \l_@@_rows_tl
4541 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_tmpc_tl` will be the color that we will use.

```

4542 \tl_clear_new:N \l_@@_color_tl
4543 \tl_set:Nx \l_@@_color_tl
4544 {
4545 \@@_color_index:n
4546 {
4547 \int_mod:nn
4548 { \l_@@_color_int - 1 }
4549 { \seq_count:N \l_@@_colors_seq }
4550 + 1
4551 }
4552 }
4553 \tl_if_empty:NF \l_@@_color_tl
4554 {
4555 \@@_add_to_colors_seq:xx
4556 { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
4557 { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
4558 }
4559 \int_incr:N \l_@@_color_int
4560 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4561 }
4562 }
4563 \endpgfpicture
4564 \group_end:
4565 }

```

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.

```

4566 \cs_new:Npn \@@_color_index:n #1
4567 {
4568 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
4569 { \@@_color_index:n { #1 - 1 } }
4570 { \seq_item:Nn \l_@@_colors_seq { #1 } }
4571 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

4572 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4573 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

4574 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
4575 {
4576   \int_compare:nNnT { #3 } > \l_tmpb_int
4577   { \int_set:Nn \l_tmpb_int { #3 } }
4578 }

4579 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
4580 {
4581   \bool_lazy_or:nnTF
4582   { \int_compare_p:nNn { #4 } = \c_zero_int }
4583   { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
4584   \prg_return_false:
4585   \prg_return_true:
4586 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

4587 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
4588 {
4589   \bool_if:nTF
4590   {
4591     \int_compare_p:n { #1 <= \l_tmpa_int }
4592     &&
4593     \int_compare_p:n { \l_tmpa_int <= #3 }
4594   }
4595   \prg_return_true:
4596   \prg_return_false:
4597 }

```

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`).

```

4598 \cs_new_protected:Npn \@@_cartesian_path:n #1
4599 {
4600   \bool_lazy_and:nnT
4601   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4602   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4603   {
4604     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4605     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4606   }

```

We begin the loop over the columns.

```

4607 \clist_map_inline:Nn \l_@@_cols_tl
4608 {
4609   \tl_set:Nn \l_tmpa_tl { ##1 }
4610   \tl_if_in:NnTF \l_tmpa_tl { - }
4611   { \@@_cut_on_hyphen:w ##1 \q_stop }
4612   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4613   \bool_lazy_or:nnT
4614   { \tl_if_blank_p:V \l_tmpa_tl }
4615   { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4616   { \tl_set:Nn \l_tmpa_tl { 1 } }

```

```

4617 \bool_lazy_or:nnT
4618 { \tl_if_blank_p:V \l_tmpb_tl }
4619 { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4620 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4621 \int_compare:nNnT \l_tmpb_tl > \c@jCol
4622 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_tmpc_tl` will contain the number of column.

```

4623 \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.

```

4624 \@@_qpoint:n { col - \l_tmpa_tl }
4625 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4626 { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4627 { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4628 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4629 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4630 \clist_map_inline:Nn \l_@@_rows_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   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4637   \tl_if_empty:NT \l_tmpb_tl
4638   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4639   \int_compare:nNnT \l_tmpb_tl > \c@iRow
4640   { \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`.

```

4641 \seq_if_in:NxF \l_@@_corners_cells_seq
4642 { \l_tmpa_tl - \l_tmpc_tl }
4643 {
4644   \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
4645   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4646   \@@_qpoint:n { row - \l_tmpa_tl }
4647   \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4648   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4649   \pgfpathrectanglecorners
4650   { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4651   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4652 }
4653 }
4654 }
4655 }

```

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`).

```

4656 \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 `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

4657 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4658 {
4659   \clist_set_eq:NN \l_tmpa_clist #1
4660   \clist_clear:N #1
4661   \clist_map_inline:Nn \l_tmpa_clist
4662   {

```

```

4663 \tl_set:Nn \l_tmpa_tl { ##1 }
4664 \tl_if_in:NnTF \l_tmpa_tl { - }
4665 { \@@_cut_on_hyphen:w ##1 \q_stop }
4666 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4667 \bool_lazy_or:nnT
4668 { \tl_if_blank_p:V \l_tmpa_tl }
4669 { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4670 { \tl_set:Nn \l_tmpa_tl { 1 } }
4671 \bool_lazy_or:nnT
4672 { \tl_if_blank_p:V \l_tmpb_tl }
4673 { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4674 { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4675 \int_compare:nNnT \l_tmpb_tl > #2
4676 { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4677 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4678 { \clist_put_right:Nn #1 { ###1 } }
4679 }
4680 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```

4681 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4682 {
4683   \peek_remove_spaces:n
4684   {
4685     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4686     {

```

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`).

```

4687       \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
4688       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4689     }
4690   }
4691 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the tabular.

```

4692 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4693 {
4694   \peek_remove_spaces:n
4695   {
4696     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4697     {
4698       \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4699       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4700       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4701     }
4702   }
4703 }

```

```

4704 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4705 {

```

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).

```

4706   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4707   {

```

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).

```

4708      \tl_gput_left:Nx \g_nicematrix_code_before_tl
4709      {
4710          \exp_not:N \columncolor [ #1 ]
4711          { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4712      }
4713  }
4714 }

```

The vertical rules

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`.

```

4715 \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`.

```

4716 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4717 {
4718     \int_compare:nNnTF \l_@@_first_col_int = 0
4719     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4720     {
4721         \int_compare:nNnTF \c@jCol = 0
4722         {
4723             \int_compare:nNnF \c@iRow = { -1 }
4724             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4725         }
4726         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4727     }
4728 }

```

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.

```

4729 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4730 {
4731     \int_compare:nNnF \c@iRow = 0
4732     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4733 }

```

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

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the column `#1` (that is to say on the left side). `#2` is the number of consecutive occurrences of `|`. `#3` and `#4` are the numbers of rows that define the delimitation of the horizontal rule that we have to draw. If `#4` is empty, that means that the rule extends until the last row.

```

4734 \cs_new_protected:Npn \@@_vline:nnnn #1 #2 #3 #4
4735 {

```

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).

```

4736   \int_compare:nNnT { #1 } < { \c@jCol + 2 }
4737   {
4738     \pgfpicture
4739     \@@_vline_i:nnnn { #1 } { #2 } { #3 } { #4 }
4740     \endpgfpicture
4741   }
4742 }
4743 \cs_new_protected:Npn \@@_vline_i:nnnn #1 #2 #3 #4
4744 {

```

`\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_tmppc_tl`.

```

4745   \tl_set:Nx \l_tmpb_tl { #1 }
4746   \tl_clear_new:N \l_tmppc_tl
4747   \int_step_variable:nnNn
4748   { #3 }
4749   { \tl_if_blank:nTF { #4 } { \int_use:N \c@iRow } { #4 } }
4750   \l_tmpa_tl
4751   {

```

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.

```

4752   \bool_gset_true:N \g_tmpa_bool
4753   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4754   { \@@_test_vline_in_block:nnnnn ##1 }
4755   \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4756   { \@@_test_vline_in_block:nnnnn ##1 }
4757   \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4758   { \@@_test_vline_in_stroken_block:nnnn ##1 }
4759   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
4760   \bool_if:NTF \g_tmpa_bool
4761   {
4762     \tl_if_empty:NT \l_tmppc_tl

```

We keep in memory that we have a rule to draw.

```

4763     { \tl_set_eq:NN \l_tmppc_tl \l_tmpa_tl }
4764   }
4765   {
4766     \tl_if_empty:NF \l_tmppc_tl
4767     {
4768       \@@_vline_ii:nnnn
4769       { #1 }
4770       { #2 }
4771       \l_tmppc_tl
4772       { \int_eval:n { \l_tmpa_tl - 1 } }
4773       \tl_clear:N \l_tmppc_tl
4774     }
4775   }
4776 }
4777 \tl_if_empty:NF \l_tmppc_tl
4778 {
4779   \@@_vline_ii:nnnn
4780   { #1 }
4781   { #2 }
4782   \l_tmppc_tl
4783   { \tl_if_blank:nTF { #4 } { \int_use:N \c@iRow } { #4 } }
4784   \tl_clear:N \l_tmppc_tl
4785 }
4786 }

```



```

4787 \cs_new_protected:Npn \@@_test_in_corner_v:
4788 {
4789   \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
4790   {
4791     \seq_if_in:NxT
4792     \l_@@_corners_cells_seq
4793     { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4794     { \bool_set_false:N \g_tmpa_bool }
4795   }
4796   {
4797     \seq_if_in:NxT
4798     \l_@@_corners_cells_seq
4799     { \l_tmpa_tl - \l_tmpb_tl }
4800     {
4801       \int_compare:nNnTF \l_tmpb_tl = 1
4802       { \bool_set_false:N \g_tmpa_bool }
4803       {
4804         \seq_if_in:NxT
4805         \l_@@_corners_cells_seq
4806         { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4807         { \bool_set_false:N \g_tmpa_bool }
4808       }
4809     }
4810   }
4811 }

```

#1 is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the numbers of the rows between which the rule has to be drawn.

```

4812 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
4813 {
4814   \bool_if:NTF \l_@@_dotted_bool
4815   { \@@_vline_iv:nnn { #1 } { #3 } { #4 } }
4816   { \@@_vline_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
4817 }

```

The following code is for the standard case (the rule which is drawn is a solid rule).

#1 is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the numbers of the rows between which the rule has to be drawn.

```

4818 \cs_new_protected:Npn \@@_vline_iii:nnnn #1 #2 #3 #4
4819 {
4820   \pgfrememberpicturepositiononpagetrue
4821   \pgf@relevantforpicturesizefalse
4822   \@@_qpoint:n { row - #3 }
4823   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4824   \@@_qpoint:n { col - #1 }
4825   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4826   \@@_qpoint:n { row - \@@_succ:n { #4 } }
4827   \dim_set_eq:NN \l_tmpc_dim \pgf@y
4828   \bool_lazy_all:nT
4829   {
4830     { \int_compare_p:nNn { #2 } > 1 }
4831     { \cs_if_exist_p:N \CT@drsc@ }
4832     { ! \tl_if_blank_p:V \CT@drsc@ }
4833   }
4834   {
4835     \group_begin:
4836     \CT@drsc@
4837     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4838     \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
4839     \dim_set:Nn \l_tmpd_dim
4840     { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4841     \pgfpathrectanglecorners

```

```

4842         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4843         { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4844         \pgfusepath { fill }
4845         \group_end:
4846     }
4847     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4848     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4849     \prg_replicate:nn { #2 - 1 }
4850     {
4851         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4852         \dim_sub:Nn \l_tmpb_dim \doublerulesep
4853         \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4854         \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4855     }
4856     \CT@arc@
4857     \pgfsetlinewidth { 1.1 \arrayrulewidth }
4858     \pgfsetrectcap
4859     \pgfusepathqstroke
4860 }

```

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

#1 is the number of the column; **#2** and **#3** are the numbers of the rows between which the rule has to be drawn.

```

4861 \cs_new_protected:Npn \@@_vline_iv:nnn #1 #2 #3
4862 {
4863     \pgfrememberpicturepositiononpagetrue
4864     \pgf@relevantforpicturesizefalse
4865     \@@_qpoint:n { col - #1 }
4866     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4867     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4868     \@@_qpoint:n { row - #2 }
4869     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4870     \@@_qpoint:n { row - \@@_succ:n { #3 } }
4871     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4872     \@@_draw_line:
4873 }

```

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).

```

4874 \cs_new_protected:Npn \@@_draw_vlines:
4875 {
4876     \int_step_inline:nnn
4877     {
4878         \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4879         1 2
4880     }
4881     {
4882         \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4883         { \@@_succ:n \c@jCol }
4884         \c@jCol
4885     }
4886     {
4887         \tl_if_eq:NnF \l_@@_vlines_clist { all }
4888         { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4889         { \@@_vline:nnnn { ##1 } 1 1 { } }
4890     }
4891 }

```

The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The rule will be drawn *before* the row #1. #2 is the number of consecutive occurrences of `\Hline`. #3 and #4 are numbers of columns that define the delimitation of the horizontal rule that we have to draw. If #4 is empty, that means that the rule extends until the last column.

```

4892 \cs_new_protected:Npn \@@_hline:nnnn #1 #2 #3 #4
4893 {
4894   \pgfpicture
4895   \@@_hline_i:nnnn { #1 } { #2 } { #3 } { #4 }
4896   \endpgfpicture
4897 }
4898 \cs_new_protected:Npn \@@_hline_i:nnnn #1 #2 #3 #4
4899 {

```

`\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`.

```

4900   \tl_set:Nn \l_tmpa_tl { #1 }
4901   \tl_clear_new:N \l_tmpc_tl
4902   \int_step_variable:nnNn
4903     { #3 }
4904     { \tl_if_blank:nTF { #4 } { \int_use:N \c@jCol } { #4 } }
4905     \l_tmpb_tl
4906   {

```

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.

```

4907     \bool_gset_true:N \g_tmpa_bool
4908     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4909       { \@@_test_hline_in_block:nnnn ##1 }
4910     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4911       { \@@_test_hline_in_block:nnnn ##1 }
4912     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4913       { \@@_test_hline_in_stroken_block:nnnn ##1 }
4914     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4915     \bool_if:NTF \g_tmpa_bool
4916     {
4917       \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4918       { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
4919     }
4920     {
4921       \tl_if_empty:NF \l_tmpc_tl
4922       {
4923         \@@_hline_ii:nnnn
4924         { #1 }
4925         { #2 }
4926         \l_tmpc_tl
4927         { \int_eval:n { \l_tmpb_tl - 1 } }
4928         \tl_clear:N \l_tmpc_tl
4929       }
4930     }
4931   }
4932   \tl_if_empty:NF \l_tmpc_tl
4933   {
4934     \@@_hline_ii:nnnn
4935     { #1 }
4936     { #2 }
4937     \l_tmpc_tl

```

```

4938         { \tl_if_blank:nTF { #4 } { \int_use:N \c@jCol } { #4 } }
4939     \tl_clear:N \l_tmpc_tl
4940 }
4941 }

```

```

4942 \cs_new_protected:Npn \@@_test_in_corner_h:
4943 {
4944     \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
4945     {
4946         \seq_if_in:NxT
4947             \l_@@_corners_cells_seq
4948             { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4949             { \bool_set_false:N \g_tmpa_bool }
4950     }
4951     {
4952         \seq_if_in:NxT
4953             \l_@@_corners_cells_seq
4954             { \l_tmpa_tl - \l_tmpb_tl }
4955             {
4956                 \int_compare:nNnTF \l_tmpa_tl = 1
4957                 { \bool_set_false:N \g_tmpa_bool }
4958                 {
4959                     \seq_if_in:NxT
4960                         \l_@@_corners_cells_seq
4961                         { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4962                         { \bool_set_false:N \g_tmpa_bool }
4963                 }
4964             }
4965     }
4966 }

```

```

4967 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
4968 {
4969     \bool_if:NTF \l_@@_dotted_bool
4970     { \@@_hline_iv:nnn { #1 } { #3 } { #4 } }
4971     { \@@_hline_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
4972 }

```

#1 is the number of the row; #2 is the number of horizontal rules to draw (with potentially a color between); #3 and #4 are the number of the columns between which the rule has to be drawn.

```

4973 \cs_new_protected:Npn \@@_hline_iii:nnnn #1 #2 #3 #4
4974 {
4975     \pgfrememberpicturepositiononpagetrue
4976     \pgf@relevantforpicturesizefalse
4977     \@@_qpoint:n { col - #3 }
4978     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4979     \@@_qpoint:n { row - #1 }
4980     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4981     \@@_qpoint:n { col - \@@_succ:n { #4 } }
4982     \dim_set_eq:NN \l_tmpc_dim \pgf@x
4983     \bool_lazy_all:nT
4984     {
4985         { \int_compare_p:nNn { #2 } > 1 }
4986         { \cs_if_exist_p:N \CT@drsc@ }
4987         { ! \tl_if_blank_p:V \CT@drsc@ }
4988     }
4989     {
4990         \group_begin:
4991         \CT@drsc@
4992         \dim_set:Nn \l_tmpd_dim
4993             { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }

```

```

4994     \pgfpathrectanglecorners
4995     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4996     { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4997     \pgfusepathqfill
4998     \group_end:
4999 }
5000 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5001 \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5002 \prg_replicate:nn { #2 - 1 }
5003 {
5004     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5005     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5006     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5007     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5008 }
5009 \CT@arc@
5010 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5011 \pgfsetrectcap
5012 \pgfusepathqstroke
5013 }

```

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}

```

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

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}

```

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

```

5014 \cs_new_protected:Npn \@@_hline_iv:nnn #1 #2 #3
5015 {
5016     \pgfrememberpicturepositiononpagetrue
5017     \pgf@relevantforpicturesizefalse
5018     \@@_qpoint:n { row - #1 }
5019     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5020     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5021     \@@_qpoint:n { col - #2 }
5022     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5023     \int_compare:nNnT { #2 } = 1
5024     {
5025         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5026         \bool_if:NT \l_@@_NiceArray_bool
5027         { \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.

```

5028     \tl_if_eq:NnF \g_@@_left_delim_tl (
5029     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
5030     )
5031     \@@_qpoint:n { col - \@@_succ:n { #3 } }
5032     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5033     \int_compare:nNnT { #3 } = \c@jCol
5034     {

```

```

5035 \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5036 \bool_if:NT \l_@@_NiceArray_bool
5037 { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5038 \tl_if_eq:NnF \g_@@_right_delim_tl )
5039 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
5040 }
5041 \@@_draw_line:
5042 }

```

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).

```

5043 \cs_new_protected:Npn \@@_draw_hlines:
5044 {
5045   \int_step_inline:nnn
5046   {
5047     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5048     1 2
5049   }
5050   {
5051     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5052     { \@@_succ:n \c@iRow }
5053     \c@iRow
5054   }
5055   {
5056     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5057     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5058     { \@@_hline:nnnn { ##1 } 1 1 { } }
5059   }
5060 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5061 \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.

```

5062 \cs_set:Npn \@@_Hline_i:n #1
5063 {
5064   \peek_meaning_ignore_spaces:NTF \Hline
5065   { \@@_Hline_ii:nn { #1 + 1 } }
5066   { \@@_Hline_iii:n { #1 } }
5067 }
5068 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5069 \cs_set:Npn \@@_Hline_iii:n #1
5070 {
5071   \skip_vertical:n
5072   {
5073     \arrayrulewidth * ( #1 )
5074     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
5075   }
5076   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5077   { \@@_hline:nnnn { \@@_succ:n { \c@iRow } } { #1 } 1 { } }
5078   \ifnum 0 = ` { \fi }
5079 }

```

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`.

```

5080 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4
5081 {
5082   \bool_lazy_all:nT
5083   {
5084     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
5085     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5086     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5087     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5088   }
5089   { \bool_gset_false:N \g_tmpa_bool }
5090 }

```

The same for vertical rules.

```

5091 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4
5092 {
5093   \bool_lazy_all:nT
5094   {
5095     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5096     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5097     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
5098     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5099   }
5100   { \bool_gset_false:N \g_tmpa_bool }
5101 }
5102 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
5103 {
5104   \bool_lazy_all:nT
5105   {
5106     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5107     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
5108     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5109     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5110   }
5111   { \bool_gset_false:N \g_tmpa_bool }
5112 }
5113 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
5114 {
5115   \bool_lazy_all:nT
5116   {
5117     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5118     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5119     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5120     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
5121   }
5122   { \bool_gset_false:N \g_tmpa_bool }
5123 }

```

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.

```

5124 \cs_new_protected:Npn \@@_compute_corners:
5125 {

```

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.

```

5126   \seq_clear_new:N \l_@@_corners_cells_seq
5127   \clist_map_inline:Nn \l_@@_corners_clist
5128   {
5129     \str_case:nnF { ##1 }

```

```

5130     {
5131         { NW }
5132         { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
5133         { NE }
5134         { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
5135         { SW }
5136         { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
5137         { SE }
5138         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
5139     }
5140     { \@@_error:nn { bad~corner } { ##1 } }
5141 }

```

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.

```

5142     \seq_if_empty:NF \l_@@_corners_cells_seq
5143     {

```

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.

```

5144         \tl_gput_right:Nx \g_@@_aux_tl
5145         {
5146             \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
5147             { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
5148         }
5149     }
5150 }

```

“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.

```

5151 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
5152 {

```

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.

```

5153     \bool_set_false:N \l_tmpa_bool
5154     \int_zero_new:N \l_@@_last_empty_row_int
5155     \int_set:Nn \l_@@_last_empty_row_int { #1 }
5156     \int_step_inline:nnnn { #1 } { #3 } { #5 }
5157     {
5158         \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
5159         \bool_lazy_or:nnTF
5160         {
5161             \cs_if_exist_p:c
5162             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
5163         }
5164         \l_tmpb_bool
5165         { \bool_set_true:N \l_tmpa_bool }
5166         {
5167             \bool_if:NF \l_tmpa_bool

```



```

5168         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
5169     }
5170 }

```

Now, you determine the last empty cell in the row of number 1.

```

5171 \bool_set_false:N \l_tmpa_bool
5172 \int_zero_new:N \l_@@_last_empty_column_int
5173 \int_set:Nn \l_@@_last_empty_column_int { #2 }
5174 \int_step_inline:nnnn { #2 } { #4 } { #6 }
5175 {
5176     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
5177     \bool_lazy_or:nnTF
5178         \l_tmpb_bool
5179         {
5180             \cs_if_exist_p:c
5181                 { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
5182         }
5183     { \bool_set_true:N \l_tmpa_bool }
5184     {
5185         \bool_if:NF \l_tmpa_bool
5186         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
5187     }
5188 }

```

Now, we loop over the rows.

```

5189 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
5190 {

```

We treat the row number ##1 with another loop.

```

5191     \bool_set_false:N \l_tmpa_bool
5192     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
5193     {
5194         \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
5195         \bool_lazy_or:nnTF
5196             \l_tmpb_bool
5197             {
5198                 \cs_if_exist_p:c
5199                     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
5200             }
5201         { \bool_set_true:N \l_tmpa_bool }
5202         {
5203             \bool_if:NF \l_tmpa_bool
5204             {
5205                 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
5206                 \seq_put_right:Nn
5207                     \l_@@_corners_cells_seq
5208                     { ##1 - #####1 }
5209             }
5210         }
5211     }
5212 }
5213 }

```

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`).

```

5214 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
5215 {
5216     \int_set:Nn \l_tmpa_int { #1 }
5217     \int_set:Nn \l_tmpb_int { #2 }
5218     \bool_set_false:N \l_tmpb_bool
5219     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5220         { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
5221 }

```

```

5222 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
5223 {
5224   \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
5225   {
5226     \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
5227     {
5228       \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
5229       {
5230         \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
5231         { \bool_set_true:N \l_tmpb_bool }
5232       }
5233     }
5234   }
5235 }

```

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`.

```

5236 \cs_new:Npn \@@_hdottedline:
5237 {
5238   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
5239   \@@_hdottedline_i:
5240 }

```

On the other side, the following command should be protected.

```

5241 \cs_new_protected:Npn \@@_hdottedline_i:
5242 {

```

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).

```

5243   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5244   { \@@_hdottedline:n { \int_use:N \c@iRow } }
5245 }

```

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.

```

5246 \cs_new_protected:Npn \@@_hdottedline:n #1
5247 {
5248   \group_begin:
5249   \bool_set_true:N \l_@@_dotted_bool
5250   \@@_hline:nnnn { #1 } { 1 } { 1 } { \int_use:N \c@jCol }
5251   \group_end:
5252 }

```

Vertical dotted lines

```

5253 \cs_new_protected:Npn \@@_vdottedline:n #1
5254 {
5255   \group_begin:
5256   \bool_set_true:N \l_@@_dotted_bool
5257   \@@_vline:nnnn { \int_eval:n { #1 + 1 } } { 1 } { 1 } { \int_use:N \c@iRow }
5258   \group_end:
5259 }

```

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.

```
5260 \bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```
5261 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5262 {
5263   auto-columns-width .code:n =
5264   {
5265     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5266     \dim_gzero_new:N \g_@@_max_cell_width_dim
5267     \bool_set_true:N \l_@@_auto_columns_width_bool
5268   }
5269 }

5270 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
5271 {
5272   \int_gincr:N \g_@@_NiceMatrixBlock_int
5273   \dim_zero:N \l_@@_columns_width_dim
5274   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5275   \bool_if:NT \l_@@_block_auto_columns_width_bool
5276   {
5277     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5278     {
5279       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
5280       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5281     }
5282   }
5283 }
```

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).

```
5284 {
5285   \bool_if:NT \l_@@_block_auto_columns_width_bool
5286   {
5287     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5288     \iow_shipout:Nx \@mainaux
5289     {
5290       \cs_gset:cpn
5291       { @@_max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```
5292       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5293     }
5294     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5295   }
5296 }
```

The extra nodes

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

```
5297 \cs_generate_variant:Nn \dim_min:nn { v n }
5298 \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).

```

5299 \cs_new_protected:Npn \@@_create_extra_nodes:
5300 {
5301   \bool_if:NTF \l_@@_medium_nodes_bool
5302   {
5303     \bool_if:NTF \l_@@_large_nodes_bool
5304     \@@_create_medium_and_large_nodes:
5305     \@@_create_medium_nodes:
5306   }
5307   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5308 }

```

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`.

```

5309 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5310 {
5311   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5312   {
5313     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
5314     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
5315     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
5316     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
5317   }
5318   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5319   {
5320     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
5321     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
5322     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
5323     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
5324   }

```

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

```

5325   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5326   {
5327     \int_step_variable:nnNn
5328     \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.

```

5329     {
5330       \cs_if_exist:cT
5331       { 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`.

```

5332     {
5333       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5334       \dim_set:cn { l_@@_row\_@@_i: _min_dim}

```

```

5335         { \dim_min:vn { l_@@_row _ @@_i: _min_dim } \pgf@y }
5336     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { @@_i: - @@_j: }
5337     {
5338         \dim_set:cn { l_@@_column _ @@_j: _min_dim}
5339         { \dim_min:vn { l_@@_column _ @@_j: _min_dim } \pgf@x }
5340     }

```

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`.

```

5341     \pgfpointanchor { @@_env: - @@_i: - @@_j: } { north-east }
5342     \dim_set:cn { l_@@_row _ @@_i: _ max_dim }
5343     { \dim_max:vn { l_@@_row _ @@_i: _ max_dim } \pgf@y }
5344     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { @@_i: - @@_j: }
5345     {
5346         \dim_set:cn { l_@@_column _ @@_j: _ max_dim }
5347         { \dim_max:vn { l_@@_column _ @@_j: _ max_dim } \pgf@x }
5348     }
5349 }
5350 }
5351 }

```

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

```

5352     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int @@_i:
5353     {
5354         \dim_compare:nNnT
5355         { \dim_use:c { l_@@_row _ @@_i: _ min _ dim } } = \c_max_dim
5356         {
5357             @@_qpoint:n { row - @@_i: - base }
5358             \dim_set:cn { l_@@_row _ @@_i: _ max _ dim } \pgf@y
5359             \dim_set:cn { l_@@_row _ @@_i: _ min _ dim } \pgf@y
5360         }
5361     }
5362     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int @@_j:
5363     {
5364         \dim_compare:nNnT
5365         { \dim_use:c { l_@@_column _ @@_j: _ min _ dim } } = \c_max_dim
5366         {
5367             @@_qpoint:n { col - @@_j: }
5368             \dim_set:cn { l_@@_column _ @@_j: _ max _ dim } \pgf@y
5369             \dim_set:cn { l_@@_column _ @@_j: _ min _ dim } \pgf@y
5370         }
5371     }
5372 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

5373 \cs_new_protected:Npn \@@_create_medium_nodes:
5374 {
5375     \pgfpicture
5376     \pgfrememberpicturepositiononpagetrue
5377     \pgf@relevantforpicturesizefalse
5378     \@@_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”.

```

5379     \tl_set:Nn \l_@@_suffix_tl { -medium }
5380     \@@_create_nodes:
5381     \endpgfpicture
5382 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁷². 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:`.

```

5383 \cs_new_protected:Npn \@@_create_large_nodes:
5384 {
5385   \pgfpicture
5386     \pgfrememberpicturepositiononpagetrue
5387     \pgf@relevantforpicturesizefalse
5388     \@@_computations_for_medium_nodes:
5389     \@@_computations_for_large_nodes:
5390     \tl_set:Nn \l_@@_suffix_tl { - large }
5391     \@@_create_nodes:
5392   \endpgfpicture
5393 }

5394 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5395 {
5396   \pgfpicture
5397     \pgfrememberpicturepositiononpagetrue
5398     \pgf@relevantforpicturesizefalse
5399     \@@_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”.

```

5400   \tl_set:Nn \l_@@_suffix_tl { - medium }
5401   \@@_create_nodes:
5402   \@@_computations_for_large_nodes:
5403   \tl_set:Nn \l_@@_suffix_tl { - large }
5404   \@@_create_nodes:
5405   \endpgfpicture
5406 }

```

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.

```

5407 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5408 {
5409   \int_set:Nn \l_@@_first_row_int 1
5410   \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`.

```

5411   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5412   {
5413     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5414     {
5415       (
5416         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5417         \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5418       )
5419       / 2
5420     }
5421     \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5422     { l_@@_row _ \@@_i: _ min _ dim }
5423   }
5424   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5425   {
5426     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5427     {

```

⁷²If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

5428      (
5429          \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5430          \dim_use:c
5431              { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5432      )
5433      / 2
5434  }
5435  \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5436  { l_@@_column _ \@@_j: _ max _ dim }
5437  }

```

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

```

5438      \dim_sub:cn
5439      { l_@@_column _ 1 _ min _ dim }
5440      \l_@@_left_margin_dim
5441      \dim_add:cn
5442      { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5443      \l_@@_right_margin_dim
5444  }

```

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).

```

5445 \cs_new_protected:Npn \@@_create_nodes:
5446 {
5447     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5448     {
5449         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5450         {

```

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

```

5451         \@@_pgf_rect_node:nnnnn
5452         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5453         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
5454         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
5455         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
5456         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
5457         \str_if_empty:NF \l_@@_name_str
5458         {
5459             \pgfnodealias
5460             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5461             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5462         }
5463     }
5464 }

```

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 .

```

5465     \seq_mapthread_function:NNN
5466     \g_@@_multicolumn_cells_seq
5467     \g_@@_multicolumn_sizes_seq
5468     \@@_node_for_multicolumn:nn
5469 }

```

```

5470 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5471 {
5472     \cs_set_nopar:Npn \@@_i: { #1 }
5473     \cs_set_nopar:Npn \@@_j: { #2 }
5474 }

```

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”).

```

5475 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5476 {
5477   \@@_extract_coords_values: #1 \q_stop
5478   \@@_pgf_rect_node:nnnnn
5479     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5480     { \dim_use:c { \l_@@_column _ \@@_j: _ min _ dim } }
5481     { \dim_use:c { \l_@@_row _ \@@_i: _ min _ dim } }
5482     { \dim_use:c { \l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
5483     { \dim_use:c { \l_@@_row _ \@@_i: _ max _ dim } }
5484   \str_if_empty:NF \l_@@_name_str
5485   {
5486     \pgfnodealias
5487       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5488       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
5489   }
5490 }

```

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.

```

5491 \keys_define:nn { NiceMatrix / Block / FirstPass }
5492 {
5493   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5494   l .value_forbidden:n = true ,
5495   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5496   r .value_forbidden:n = true ,
5497   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5498   c .value_forbidden:n = true ,
5499   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5500   L .value_forbidden:n = true ,
5501   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5502   R .value_forbidden:n = true ,
5503   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5504   C .value_forbidden:n = true ,
5505   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5506   t .value_forbidden:n = true ,
5507   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5508   b .value_forbidden:n = true ,
5509   color .tl_set:N = \l_@@_color_tl ,
5510   color .value_required:n = true
5511 }

```

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.

```

5512 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
5513 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not be provided by the user, you use 1-1 (that is to say a block of only one cell).

```

5514   \peek_remove_spaces:n
5515   {
5516     \tl_if_blank:nTF { #2 }

```



```

5517         { \@@_Block_i 1-1 \q_stop }
5518         { \@@_Block_i #2 \q_stop }
5519     { #1 } { #3 } { #4 }
5520 }
5521 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5522 \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, $\#4$ 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.

```

5523 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5524 {

```

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).

```

5525     \bool_lazy_or:nnTF
5526     { \tl_if_blank_p:n { #1 } }
5527     { \str_if_eq_p:nn { #1 } { * } }
5528     { \int_set:Nn \l_tmpa_int { 100 } }
5529     { \int_set:Nn \l_tmpa_int { #1 } }
5530     \bool_lazy_or:nnTF
5531     { \tl_if_blank_p:n { #2 } }
5532     { \str_if_eq_p:nn { #2 } { * } }
5533     { \int_set:Nn \l_tmpb_int { 100 } }
5534     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5535     \int_compare:nNnTF \l_tmpb_int = 1
5536     {
5537         \str_if_empty:NTF \l_@@_hpos_cell_str
5538         { \str_set:Nn \l_@@_hpos_block_str c }
5539         { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
5540     }
5541     { \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.

```

5542     \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
5543     \tl_set:Nx \l_tmpa_tl
5544     {
5545         { \int_use:N \c@iRow }
5546         { \int_use:N \c@jCol }
5547         { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5548         { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5549     }

```

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 currying).

```

5550     \bool_if:nTF
5551     {
5552     (

```

```

5553     \int_compare_p:nNn { \l_tmpa_int } = 1
5554     ||
5555     \int_compare_p:nNn { \l_tmpb_int } = 1
5556   )
5557   && ! \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.

```

5558     && ! \l_@@_X_column_bool
5559   }
5560   { \exp_args:Nxx \@@_Block_iv:nnnnn }
5561   { \exp_args:Nxx \@@_Block_v:nnnnn }
5562   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5563 }

```

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).

```

5564 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5565 {
5566   \int_gincr:N \g_@@_block_box_int
5567   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5568   {
5569     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5570     {
5571       \@@_actually_diagbox:nnnnnn
5572       { \int_use:N \c@iRow }
5573       { \int_use:N \c@jCol }
5574       { \int_eval:n { \c@iRow + #1 - 1 } }
5575       { \int_eval:n { \c@jCol + #2 - 1 } }
5576       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5577     }
5578   }
5579   \box_gclear_new:c
5580   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
5581   \hbox_gset:cn
5582   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
5583   {

```

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}**).

```

5584     \tl_if_empty:NTF \l_@@_color_tl
5585     { \int_compare:nNnT { #2 } = 1 \set@color }
5586     { \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**.

```

5587     \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5588     \group_begin:
5589     \cs_set:Npn \arraystretch { 1 }
5590     \dim_zero:N \extrarowheight
5591     #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).

```

5592     \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
5593     \bool_if:NTF \l_@@_NiceTabular_bool
5594     {
5595         \bool_lazy_and:nnTF
5596         { \int_compare_p:nNn { #2 } = 1 }
5597         { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }

```

When the block is mono-column in a column with a fixed width (eg p{3cm}).

```

5598     {
5599         \begin { minipage } [ \l_@@_vpos_of_block_tl ]
5600         { \l_@@_col_width_dim }
5601         \str_case:Vn \l_@@_hpos_block_str
5602         {
5603             c \centering
5604             r \raggedleft
5605             l \raggedright
5606         }
5607         #5
5608         \end { minipage }
5609     }
5610     {
5611         \use:x
5612         {
5613             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5614             { @ { } \l_@@_hpos_block_str @ { } }
5615         }
5616         #5
5617         \end { tabular }
5618     }
5619 }
5620 {
5621     \c_math_toggle_token
5622     \use:x
5623     {
5624         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5625         { @ { } \l_@@_hpos_block_str @ { } }
5626     }
5627     #5
5628     \end { array }
5629     \c_math_toggle_token
5630 }
5631 \group_end:
5632 }
5633 \bool_if:NT \g_@@_rotate_bool
5634 {
5635     \box_grotate:cn
5636     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5637     { 90 }
5638     \bool_gset_false:N \g_@@_rotate_bool
5639 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5640     \int_compare:nNnT { #2 } = 1
5641     {
5642         \dim_gset:Nn \g_@@_blocks_wd_dim
5643         {
5644             \dim_max:nn
5645             \g_@@_blocks_wd_dim
5646             {
5647                 \box_wd:c

```

```

5648         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5649     }
5650 }
5651 }

```

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.

```

5652 \int_compare:nNnT { #1 } = 1
5653 {
5654     \dim_gset:Nn \g_@@_blocks_ht_dim
5655     {
5656         \dim_max:nn
5657         \g_@@_blocks_ht_dim
5658         {
5659             \box_ht:c
5660             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5661         }
5662     }
5663     \dim_gset:Nn \g_@@_blocks_dp_dim
5664     {
5665         \dim_max:nn
5666         \g_@@_blocks_dp_dim
5667         {
5668             \box_dp:c
5669             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5670         }
5671     }
5672 }
5673 \seq_gput_right:Nx \g_@@_blocks_seq
5674 {
5675     \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.

```

5676     { \exp_not:n { #3 } , \l_@@_hpos_block_str }
5677     {
5678         \box_use_drop:c
5679         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5680     }
5681 }
5682 }

```

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.

```

5683 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5684 {
5685     \seq_gput_right:Nx \g_@@_blocks_seq
5686     {
5687         \l_tmpa_tl
5688         { \exp_not:n { #3 } }
5689         \exp_not:n
5690         {
5691             {
5692                 \bool_if:NTF \l_@@_NiceTabular_bool
5693                 {
5694                     \group_begin:
5695                     \cs_set:Npn \arraystretch { 1 }
5696                     \dim_zero:N \extrarowheight
5697                     #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).

```

5698         \bool_if:NT \g_@@_rotate_bool
5699         { \str_set:Nn \l_@@_hpos_block_str c }
5700     \use:x
5701     {
5702         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5703         { @ { } \l_@@_hpos_block_str @ { } }
5704     }
5705     #5
5706     \end { tabular }
5707     \group_end:
5708 }
5709 {
5710     \group_begin:
5711     \cs_set:Npn \arraystretch { 1 }
5712     \dim_zero:N \extrarowheight
5713     #4
5714     \bool_if:NT \g_@@_rotate_bool
5715     { \str_set:Nn \l_@@_hpos_block_str c }
5716     \c_math_toggle_token
5717     \use:x
5718     {
5719         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5720         { @ { } \l_@@_hpos_block_str @ { } }
5721     }
5722     #5
5723     \end { array }
5724     \c_math_toggle_token
5725     \group_end:
5726 }
5727 }
5728 }
5729 }
5730 }
```

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).

```

5731 \keys_define:nn { NiceMatrix / Block / SecondPass }
5732 {
5733     tikz .code:n =
5734         \bool_if:NTF \c_@@_tikz_loaded_bool
5735         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
5736         { \@@_error:n { tikz-key-without~tikz } } ,
5737     tikz .value_required:n = true ,
5738     fill .tl_set:N = \l_@@_fill_tl ,
5739     fill .value_required:n = true ,
5740     draw .tl_set:N = \l_@@_draw_tl ,
5741     draw .default:n = default ,
5742     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5743     rounded-corners .default:n = 4 pt ,
5744     color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5745     color .value_required:n = true ,
5746     borders .clist_set:N = \l_@@_borders_clist ,
5747     borders .value_required:n = true ,
5748     hvlines .bool_set:N = \l_@@_hvlines_block_bool ,
5749     hvlines .default:n = true ,
5750     line-width .dim_set:N = \l_@@_line_width_dim ,
```

```

5751 line-width .value_required:n = true ,
5752 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5753 l .value_forbidden:n = true ,
5754 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5755 r .value_forbidden:n = true ,
5756 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5757 c .value_forbidden:n = true ,
5758 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
5759     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5760 L .value_forbidden:n = true ,
5761 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
5762     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5763 R .value_forbidden:n = true ,
5764 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
5765     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5766 C .value_forbidden:n = true ,
5767 t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5768 t .value_forbidden:n = true ,
5769 b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5770 b .value_forbidden:n = true ,
5771 name .tl_set:N = \l_@@_block_name_str ,
5772 name .value_required:n = true ,
5773 name .initial:n = \c_empty_tl ,
5774 unknown .code:n = \@@_error:n { Unknown-key-for-Block }
5775 }

```

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.

```

5776 \cs_new_protected:Npn \@@_draw_blocks:
5777 {
5778   \cs_set_eq:NN \ialign \@@_old_ialign:
5779   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
5780 }
5781 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5782 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5783   \int_zero_new:N \l_@@_last_row_int
5784   \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.

```

5785   \int_compare:nNnTF { #3 } > { 99 }
5786   { \int_set_eq:NN \l_@@_last_row_int \c_iRow }
5787   { \int_set:Nn \l_@@_last_row_int { #3 } }
5788   \int_compare:nNnTF { #4 } > { 99 }
5789   { \int_set_eq:NN \l_@@_last_col_int \c_jCol }
5790   { \int_set:Nn \l_@@_last_col_int { #4 } }
5791   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
5792   {
5793     \int_compare:nTF
5794     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
5795     {
5796       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
5797       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
5798       \group_begin:

```

```

5799         \globaldefs = 1
5800         \@@_msg_redirect_name:nn { columns~not~used } { none }
5801         \group_end:
5802     }
5803     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5804 }
5805 {
5806     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
5807     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5808     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
5809 }
5810 }
5811 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
5812 {

```

The group is for the keys.

```

5813     \group_begin:
5814     \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
5815     \bool_if:NTF \l_@@_hvlines_block_bool
5816     {
5817         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5818         {
5819             \@@_hvlines_block:nnn
5820             { \exp_not:n { #5 } }
5821             { #1 - #2 }
5822             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5823         }
5824     }
5825     {

```

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).

```

5826         \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
5827         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
5828     }
5829     \tl_if_empty:NF \l_@@_draw_tl
5830     {
5831         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5832         {
5833             \@@_stroke_block:nnn
5834             { \exp_not:n { #5 } }
5835             { #1 - #2 }
5836             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5837         }
5838         \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
5839         { { #1 } { #2 } { #3 } { #4 } }
5840     }
5841     \clist_if_empty:NF \l_@@_borders_clist
5842     {
5843         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5844         {
5845             \@@_stroke_borders_block:nnn
5846             { \exp_not:n { #5 } }
5847             { #1 - #2 }
5848             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5849         }
5850     }
5851     \tl_if_empty:NF \l_@@_fill_tl
5852     {

```

The command \@@_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.

```

5853 \exp_last_unbraced:N \l_@@_extract_brackets \l_@@_fill_tl \q_stop
5854 \tl_gput_right:Nx \g_nicematrix_code_before_tl
5855 {
5856   \exp_not:N \roundedrectanglecolor
5857   [ \l_tmpa_tl ]
5858   { \exp_not:N \l_tmpb_tl }
5859   { #1 - #2 }
5860   { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5861   { \dim_use:N \l_@@_rounded_corners_dim }
5862 }
5863 }
5864 \seq_if_empty:NF \l_@@_tikz_seq
5865 {
5866   \tl_gput_right:Nx \g_nicematrix_code_before_tl
5867   {
5868     \@@_block_tikz:nnnnn
5869     { #1 }
5870     { #2 }
5871     { \int_use:N \l_@@_last_row_int }
5872     { \int_use:N \l_@@_last_col_int }
5873     { \seq_use:Nn \l_@@_tikz_seq { , } }
5874   }
5875 }
5876 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5877 {
5878   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5879   {
5880     \@@_actually_diagbox:nnnnnn
5881     { #1 }
5882     { #2 }
5883     { \int_use:N \l_@@_last_row_int }
5884     { \int_use:N \l_@@_last_col_int }
5885     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5886   }
5887 }
5888 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
5889 \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.

```

5890 \pgfpicture

```



```

5891 \pgfrememberpicturepositiononpagetrue
5892 \pgf@relevantforpicturesizefalse
5893 \@@_qpoint:n { row - #1 }
5894 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5895 \@@_qpoint:n { col - #2 }
5896 \dim_set_eq:NN \l_tmpb_dim \pgf@x
5897 \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
5898 \dim_set_eq:NN \l_tmpc_dim \pgf@y
5899 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5900 \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.

```

5901 \@@_pgf_rect_node:nnnnn
5902 { \@@_env: - #1 - #2 - block }
5903 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
5904 \str_if_empty:NF \l_@@_block_name_str
5905 {
5906   \pgfnodealias
5907   { \@@_env: - \l_@@_block_name_str }
5908   { \@@_env: - #1 - #2 - block }
5909   \str_if_empty:NF \l_@@_name_str
5910   {
5911     \pgfnodealias
5912     { \l_@@_name_str - \l_@@_block_name_str }
5913     { \@@_env: - #1 - #2 - block }
5914   }
5915 }

```

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.

```

5916 \bool_if:NF \l_@@_hpos_of_block_cap_bool
5917 {
5918   \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.

```

5919 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5920 {

```

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.

```

5921 \cs_if_exist:cT
5922 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5923 {
5924   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5925   {
5926     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
5927     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
5928   }
5929 }
5930 }

```

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.

```

5931 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
5932 {
5933   \@@_qpoint:n { col - #2 }
5934   \dim_set_eq:NN \l_tmpb_dim \pgf@x
5935 }
5936 \dim_set:Nn \l_tmpd_dim { - \c_max_dim }

```

```

5937 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5938 {
5939   \cs_if_exist:cT
5940   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5941   {
5942     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5943     {
5944       \pgfpointanchor
5945       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5946       { east }
5947       \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
5948     }
5949   }
5950 }
5951 \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
5952 {
5953   \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5954   \dim_set_eq:NN \l_tmpd_dim \pgf@x
5955 }
5956 \@@_pgf_rect_node:nnnnn
5957 { \@@_env: - #1 - #2 - block - short }
5958 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
5959 }

```

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.

```

5960 \bool_if:NT \l_@@_medium_nodes_bool
5961 {
5962   \@@_pgf_rect_node:nnn
5963   { \@@_env: - #1 - #2 - block - medium }
5964   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
5965   {
5966     \pgfpointanchor
5967     { \@@_env:
5968       - \int_use:N \l_@@_last_row_int
5969       - \int_use:N \l_@@_last_col_int - medium
5970     }
5971     { south-east }
5972   }
5973 }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

5974 \int_compare:nNnTF { #1 } = { #3 }
5975 {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

5976 \int_compare:nNnTF { #1 } = 0
5977 { \l_@@_code_for_first_row_tl }
5978 {
5979   \int_compare:nNnT { #1 } = \l_@@_last_row_int
5980   \l_@@_code_for_last_row_tl
5981 }

```

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`.

```

5982 \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

5983 \pgfpointanchor
5984 {
5985   \@@_env: - #1 - #2 - block
5986   \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
5987 }

```

```

5988     {
5989         \str_case:Vn \l_@@_hpos_block_str
5990         {
5991             c { center }
5992             l { west }
5993             r { east }
5994         }
5995     }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

5996     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
5997     \pgfset { inner-sep = \c_zero_dim }
5998     \pgfnode
5999     { rectangle }
6000     {
6001         \str_case:Vn \l_@@_hpos_block_str
6002         {
6003             c { base }
6004             l { base~west }
6005             r { base~east }
6006         }
6007     }
6008     { \box_use_drop:N \l_@@_cell_box } { } { }
6009 }

```

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).

```

6010     {

```

If we are in the first column, we must put the block as if it was with the key r.

```

6011     \int_compare:nNnT { #2 } = 0
6012     { \str_set:Nn \l_@@_hpos_block_str r }
6013     \bool_if:nT \g_@@_last_col_found_bool
6014     {
6015         \int_compare:nNnT { #2 } = \g_@@_col_total_int
6016         { \str_set:Nn \l_@@_hpos_block_str l }
6017     }
6018     \pgftransformshift
6019     {
6020         \pgfpointanchor
6021         {
6022             \@@_env: - #1 - #2 - block
6023             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6024         }
6025         {
6026             \str_case:Vn \l_@@_hpos_block_str
6027             {
6028                 c { center }
6029                 l { west }
6030                 r { east }
6031             }
6032         }
6033     }
6034     \pgfset { inner-sep = \c_zero_dim }
6035     \pgfnode
6036     { rectangle }
6037     {
6038         \str_case:Vn \l_@@_hpos_block_str
6039         {
6040             c { center }
6041             l { west }
6042             r { east }
6043         }
6044     }

```

```

6045         { \box_use_drop:N \l_@@_cell_box } { } { }
6046     }
6047     \endpgfpicture
6048     \group_end:
6049 }

6050 \NewDocumentCommand \@@_extract_brackets { 0 { } }
6051 {
6052     \tl_set:Nn \l_tmpa_tl { #1 }
6053     \@@_store_in_tmpb_tl
6054 }
6055 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
6056 { \tl_set:Nn \l_tmpb_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).

```

6057 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
6058 {
6059     \group_begin:
6060     \tl_clear:N \l_@@_draw_tl
6061     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6062     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
6063     \pgfpicture
6064     \pgfrememberpicturepositiononpagetrue
6065     \pgf@relevantforpicturesizefalse
6066     \tl_if_empty:NF \l_@@_draw_tl
6067     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

6068         \str_if_eq:VnTF \l_@@_draw_tl { default }
6069         { \CT@arc@ }
6070         { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
6071     }
6072     \pgfsetcornersarced
6073     {
6074         \pgfpoint
6075         { \dim_use:N \l_@@_rounded_corners_dim }
6076         { \dim_use:N \l_@@_rounded_corners_dim }
6077     }
6078     \@@_cut_on_hyphen:w #2 \q_stop
6079     \bool_lazy_and:nnT
6080     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
6081     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
6082     {
6083         \@@_qpoint:n { row - \l_tmpa_tl }
6084         \dim_set:Nn \l_tmpb_dim { \pgf@y }
6085         \@@_qpoint:n { col - \l_tmpb_tl }
6086         \dim_set:Nn \l_tmpc_dim { \pgf@x }
6087         \@@_cut_on_hyphen:w #3 \q_stop
6088         \int_compare:nNnT \l_tmpa_tl > \c@iRow
6089         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
6090         \int_compare:nNnT \l_tmpb_tl > \c@jCol
6091         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
6092         \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
6093         \dim_set:Nn \l_tmpa_dim { \pgf@y }
6094         \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
6095         \dim_set:Nn \l_tmpd_dim { \pgf@x }
6096         \pgfpathrectanglecorners
6097         { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
6098         { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
6099         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

6100     \pgfusepath { stroke }
6101   }
6102   \endpgfpicture
6103   \group_end:
6104 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

6105 \keys_define:nn { NiceMatrix / BlockStroke }
6106 {
6107   color .tl_set:N = \l_@@_draw_tl ,
6108   draw .tl_set:N = \l_@@_draw_tl ,
6109   draw .default:n = default ,
6110   line-width .dim_set:N = \l_@@_line_width_dim ,
6111   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6112   rounded-corners .default:n = 4 pt
6113 }

```

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).

```

6114 \cs_new_protected:Npn \@@_hvlines_block:nnn #1 #2 #3
6115 {
6116   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6117   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6118   \@@_cut_on_hyphen:w #2 \q_stop
6119   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6120   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6121   \@@_cut_on_hyphen:w #3 \q_stop
6122   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6123   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6124   \int_step_inline:nnn \l_tmpd_tl \l_tmpb_tl
6125   {
6126     \use:x
6127     { \@@_vline:nnnn { ##1 } 1 { \l_tmpc_tl } { \@@_pred:n \l_tmpa_tl } }
6128   }
6129   \int_step_inline:nnn \l_tmpc_tl \l_tmpa_tl
6130   {
6131     \use:x
6132     { \@@_hline:nnnn { ##1 } 1 { \l_tmpd_tl } { \@@_pred:n \l_tmpb_tl } }
6133   }
6134 }

```

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).

```

6135 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
6136 {
6137   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6138   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6139   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
6140   { \@@_error:n { borders~forbidden } }
6141   {
6142     \clist_map_inline:Nn \l_@@_borders_clist
6143     {
6144       \clist_if_in:nnF { top , bottom , left , right } { ##1 }
6145       { \@@_error:nn { bad~border } { ##1 } }
6146     }
6147     \@@_cut_on_hyphen:w #2 \q_stop
6148     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6149     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6150     \@@_cut_on_hyphen:w #3 \q_stop

```

```

6151 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6152 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6153 \pgfpicture
6154 \pgfrememberpicturerepositiononpagetrue
6155 \pgf@relevantforpicturesizefalse
6156 \CT@arc@
6157 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
6158 \clist_if_in:NnT \l_@@_borders_clist { right }
6159 { \@@_stroke_vertical:n \l_tmpb_tl }
6160 \clist_if_in:NnT \l_@@_borders_clist { left }
6161 { \@@_stroke_vertical:n \l_tmpd_tl }
6162 \clist_if_in:NnT \l_@@_borders_clist { bottom }
6163 { \@@_stroke_horizontal:n \l_tmpa_tl }
6164 \clist_if_in:NnT \l_@@_borders_clist { top }
6165 { \@@_stroke_horizontal:n \l_tmpc_tl }
6166 \endpgfpicture
6167 }
6168 }

```

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).

```

6169 \cs_new_protected:Npn \@@_stroke_vertical:n #1
6170 {
6171   \@@_qpoint:n \l_tmpc_tl
6172   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6173   \@@_qpoint:n \l_tmpa_tl
6174   \dim_set:Nn \l_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6175   \@@_qpoint:n { #1 }
6176   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
6177   \pgfpathlineto { \pgfpoint \pgf@x \l_tmpc_dim }
6178   \pgfusepathqstroke
6179 }

```

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).

```

6180 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
6181 {
6182   \@@_qpoint:n \l_tmpd_tl
6183   \clist_if_in:NnTF \l_@@_borders_clist { left }
6184   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
6185   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
6186   \@@_qpoint:n \l_tmpb_tl
6187   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
6188   \@@_qpoint:n { #1 }
6189   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
6190   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6191   \pgfusepathqstroke
6192 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

6193 \keys_define:nn { NiceMatrix / BlockBorders }
6194 {
6195   borders .clist_set:N = \l_@@_borders_clist ,
6196   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6197   rounded-corners .default:n = 4 pt ,
6198   line-width .dim_set:N = \l_@@_line_width_dim
6199 }

```

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.

```

6200 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5

```

```

6201 {
6202   \begin { tikzpicture }
6203   \clist_map_inline:nn { #5 }
6204     {
6205       \path [ ##1 ]
6206         ( #1 -| #2 ) rectangle ( \@@_succ:n { #3 } -| \@@_succ:n { #4 } ) ;
6207     }
6208   \end { tikzpicture }
6209 }

```

How to draw the dotted lines transparently

```

6210 \cs_set_protected:Npn \@@_renew_matrix:
6211 {
6212   \RenewDocumentEnvironment { pmatrix } { } {
6213     { \pNiceMatrix }
6214     { \endpNiceMatrix }
6215   \RenewDocumentEnvironment { vmatrix } { } {
6216     { \vNiceMatrix }
6217     { \endvNiceMatrix }
6218   \RenewDocumentEnvironment { Vmatrix } { } {
6219     { \VNiceMatrix }
6220     { \endVNiceMatrix }
6221   \RenewDocumentEnvironment { bmatrix } { } {
6222     { \bNiceMatrix }
6223     { \endbNiceMatrix }
6224   \RenewDocumentEnvironment { Bmatrix } { } {
6225     { \BNiceMatrix }
6226     { \endBNiceMatrix }
6227 }

```

Automatic arrays

```

6228 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
6229 {
6230   \int_set:Nn \l_@@_nb_rows_int { #1 }
6231   \int_set:Nn \l_@@_nb_cols_int { #2 }
6232 }

```

We will extract the potential keys l, r and c and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

6233 \keys_define:nn { NiceMatrix / Auto }
6234 {
6235   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
6236   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
6237   c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
6238 }
6239 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
6240 {
6241   \int_zero_new:N \l_@@_nb_rows_int
6242   \int_zero_new:N \l_@@_nb_cols_int
6243   \@@_set_size:n #4 \q_stop

```

The group is for the protection of `\l_@@_type_of_col_tl`.

```

6244   \group_begin:
6245   \tl_set:Nn \l_@@_type_of_col_tl c
6246   \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
6247   \use:x
6248   {
6249     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
6250     { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
6251     [ \exp_not:N \l_tmpa_tl ]

```

```

6252     }
6253     \int_compare:nNnT \l_@@_first_row_int = 0
6254     {
6255         \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6256         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { & }
6257         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6258     }
6259     \prg_replicate:nn \l_@@_nb_rows_int
6260     {
6261         \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).

```

6262         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
6263         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6264     }
6265     \int_compare:nNnT \l_@@_last_row_int > { -2 }
6266     {
6267         \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6268         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6269         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6270     }
6271     \end { NiceArrayWithDelims }
6272     \group_end:
6273 }
6274 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6275 {
6276     \cs_set_protected:cpn { #1 AutoNiceMatrix }
6277     {
6278         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6279         \AutoNiceMatrixWithDelims { #2 } { #3 }
6280     }
6281 }
6282 \@@_define_com:nnn p ( )
6283 \@@_define_com:nnn b [ ]
6284 \@@_define_com:nnn v | |
6285 \@@_define_com:nnn V \ | \ |
6286 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

6287 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
6288 {
6289     \group_begin:
6290     \bool_set_true:N \l_@@_NiceArray_bool
6291     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6292     \group_end:
6293 }

```

The redefinition of the command \dotfill

```

6294 \cs_set_eq:NN \@@_old_dotfill \dotfill
6295 \cs_new_protected:Npn \@@_dotfill:
6296 {

```

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}).

```

6297     \@@_old_dotfill
6298     \bool_if:NT \l_@@_NiceTabular_bool
6299     { \group_insert_after:N \@@_dotfill_ii: }
6300     { \group_insert_after:N \@@_dotfill_i: }
6301 }
6302 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }

```



```
6303 \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 its 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`.

```
6304 \cs_new_protected:Npn \@@_dotfill_iii:
6305   { \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.

```
6306 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6307   {
6308     \tl_gput_right:Nx \g_@@_internal_code_after_tl
6309     {
6310       \@@_actually_diagbox:nnnnnn
6311       { \int_use:N \c@iRow }
6312       { \int_use:N \c@jCol }
6313       { \int_use:N \c@iRow }
6314       { \int_use:N \c@jCol }
6315       { \exp_not:n { #1 } }
6316       { \exp_not:n { #2 } }
6317     }
6318   }
```

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`.

```
6318   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6319   {
6320     { \int_use:N \c@iRow }
6321     { \int_use:N \c@jCol }
6322     { \int_use:N \c@iRow }
6323     { \int_use:N \c@jCol }
6324   }
```

The last argument is for the name of the block.

```
6324   { }
6325 }
6326 }
```

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.

```
6327 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
6328   {
6329     \pgfpicture
6330     \pgf@relevantforpicturesizefalse
6331     \pgfrememberpicturepositiononpagetrue
6332     \@@_qpoint:n { row - #1 }
6333     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6334     \@@_qpoint:n { col - #2 }
6335     \dim_set_eq:NN \l_tmpb_dim \pgf@x
6336     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6337     \@@_qpoint:n { row - \@@_succ:n { #3 } }
6338     \dim_set_eq:NN \l_tmpc_dim \pgf@y
6339     \@@_qpoint:n { col - \@@_succ:n { #4 } }
6340     \dim_set_eq:NN \l_tmpd_dim \pgf@x
6341     \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
6342     {
```

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.

```

6343     \CT@arc@
6344     \pgfsetroundcap
6345     \pgfusepathqstroke
6346   }
6347   \pgfset { inner~sep = 1 pt }
6348   \pgfscope
6349   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
6350   \pgfnode { rectangle } { south-west }
6351   {
6352     \begin { minipage } { 20 cm }
6353     \@@_math_toggle_token: #5 \@@_math_toggle_token:
6354     \end { minipage }
6355   }
6356   { }
6357   { }
6358   \endpgfscope
6359   \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
6360   \pgfnode { rectangle } { north-east }
6361   {
6362     \begin { minipage } { 20 cm }
6363     \raggedleft
6364     \@@_math_toggle_token: #6 \@@_math_toggle_token:
6365     \end { minipage }
6366   }
6367   { }
6368   { }
6369   \endpgfpicture
6370 }

```

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.

```

6371 \keys_define:nn { NiceMatrix }
6372 {
6373   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6374   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6375 }
6376 \keys_define:nn { NiceMatrix / CodeAfter }
6377 {
6378   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6379   sub-matrix .value_required:n = true ,
6380   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6381   delimiters / color .value_required:n = true ,
6382   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6383   rules .value_required:n = true ,
6384   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
6385 }

```

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. 120.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

6386 \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 `\`.

```
6387 \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`.

```
6388 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
6389 {
6390   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6391   \@@_CodeAfter_iv:n
6392 }
```

We catch the argument of the command `\end` (in `#1`).

```
6393 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
6394 {
```

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.

```
6395   \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`.

```
6396   {
6397     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6398     \@@_CodeAfter_ii:n
6399   }
6400 }
```

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 columnn. 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.

```
6401 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
6402 {
6403   \pgfpicture
6404   \pgfrememberpicturepositiononpagetrue
6405   \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.

```
6406   \@@_qpoint:n { row - 1 }
6407   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6408   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
6409   \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).

```
6410   \bool_if:nTF { #3 }
6411     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6412     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6413   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6414     {
6415       \cs_if_exist:cT
```

```

6416 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6417 {
6418   \pgfpointanchor
6419   { \@@_env: - ##1 - #2 }
6420   { \bool_if:nTF { #3 } { west } { east } }
6421   \dim_set:Nn \l_tmpa_dim
6422   { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6423 }
6424 }

```

Now we can put the delimiter with a node of PGF.

```

6425 \pgfset { inner~sep = \c_zero_dim }
6426 \dim_zero:N \nulldelimiterspace
6427 \pgftransformshift
6428 {
6429   \pgfpoint
6430   { \l_tmpa_dim }
6431   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6432 }
6433 \pgfnode
6434 { rectangle }
6435 { \bool_if:nTF { #3 } { east } { west } }
6436 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

6437 \nullfont
6438 \c_math_toggle_token
6439 \tl_if_empty:NF \l_@@_delimiters_color_tl
6440 { \color { \l_@@_delimiters_color_tl } }
6441 \bool_if:nTF { #3 } { \left #1 } { \left . }
6442 \vcenter
6443 {
6444   \nullfont
6445   \hrule \@height
6446   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6447   \@depth \c_zero_dim
6448   \@width \c_zero_dim
6449 }
6450 \bool_if:nTF { #3 } { \right . } { \right #1 }
6451 \c_math_toggle_token
6452 }
6453 { }
6454 { }
6455 \endpgfpicture
6456 }

```

The command `\SubMatrix`

```

6457 \keys_define:nn { NiceMatrix / sub-matrix }
6458 {
6459   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6460   extra-height .value_required:n = true ,
6461   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6462   left-xshift .value_required:n = true ,
6463   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6464   right-xshift .value_required:n = true ,
6465   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6466   xshift .value_required:n = true ,
6467   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6468   delimiters / color .value_required:n = true ,
6469   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6470   slim .default:n = true ,
6471   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6472   hlines .default:n = all ,

```

```

6473 vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6474 vlines .default:n = all ,
6475 hvlines .meta:n = { hlines, vlines } ,
6476 hvlines .value_forbidden:n = true ,
6477 }
6478 \keys_define:nn { NiceMatrix }
6479 {
6480   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6481   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6482   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6483   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6484   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6485   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6486 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

6487 \keys_define:nn { NiceMatrix / SubMatrix }
6488 {
6489   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6490   hlines .default:n = all ,
6491   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6492   vlines .default:n = all ,
6493   hvlines .meta:n = { hlines, vlines } ,
6494   hvlines .value_forbidden:n = true ,
6495   name .code:n =
6496     \tl_if_empty:nTF { #1 }
6497     { \@@_error:n { Invalid-name-format } }
6498     {
6499       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6500       {
6501         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6502         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
6503         {
6504           \str_set:Nn \l_@@_submatrix_name_str { #1 }
6505           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6506         }
6507       }
6508       { \@@_error:n { Invalid-name-format } }
6509     } ,
6510   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6511   rules .value_required:n = true ,
6512   code .tl_set:N = \l_@@_code_tl ,
6513   code .value_required:n = true ,
6514   name .value_required:n = true ,
6515   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
6516 }
6517 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
6518 {
6519   \peek_remove_spaces:n
6520   {
6521     \@@_cut_on_hyphen:w #3 \q_stop
6522     \tl_clear_new:N \l_tmpc_tl
6523     \tl_clear_new:N \l_tmpd_tl
6524     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6525     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6526     \@@_cut_on_hyphen:w #2 \q_stop
6527     \seq_gput_right:Nx \g_@@_submatrix_seq
6528     { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
6529     \tl_gput_right:Nn \g_@@_internal_code_after_tl
6530     { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6531   }

```

6532 }

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`.

```

6533 \AtBeginDocument
6534 {
6535   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
6536   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
6537   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
6538     {
6539       \peek_remove_spaces:n
6540       {
6541         \@@_sub_matrix:nnnnnnn
6542         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
6543       }
6544     }
6545 }
```

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).

```

6546 \cs_new_protected:Npn \@@_compute_i_j:nn #1 #2
6547 {
6548   \tl_clear_new:N \l_@@_first_i_tl
6549   \tl_clear_new:N \l_@@_first_j_tl
6550   \tl_clear_new:N \l_@@_last_i_tl
6551   \tl_clear_new:N \l_@@_last_j_tl
6552   \@@_cut_on_hyphen:w #1 \q_stop
6553   \tl_if_eq:NnTF \l_tmpa_tl { last }
6554     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
6555     { \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl }
6556   \tl_if_eq:NnTF \l_tmpb_tl { last }
6557     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
6558     { \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl }
6559   \@@_cut_on_hyphen:w #2 \q_stop
6560   \tl_if_eq:NnTF \l_tmpa_tl { last }
6561     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
6562     { \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl }
6563   \tl_if_eq:NnTF \l_tmpb_tl { last }
6564     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
6565     { \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl }
6566 }
6567 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6568 {
6569   \group_begin:
```

The four following token lists correspond to the position of the `\SubMatrix`.

```

6570 \@@_compute_i_j:nn { #2 } { #3 }
6571 \bool_lazy_or:nnTF
6572 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6573 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6574 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
6575 {
6576   \str_clear_new:N \l_@@_submatrix_name_str
6577   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6578   \pgfpicture
6579   \pgfrememberpicturepositiononpagetrue
6580   \pgf@relevantforpicturesizefalse
6581   \pgfset { inner-sep = \c_zero_dim }
6582   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6583   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currying.

```

6584 \bool_if:NTF \l_@@_submatrix_slim_bool
6585 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6586 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6587 {
6588   \cs_if_exist:cT
6589   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6590   {
6591     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6592     \dim_set:Nn \l_@@_x_initial_dim
6593     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6594   }
6595   \cs_if_exist:cT
6596   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6597   {
6598     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6599     \dim_set:Nn \l_@@_x_final_dim
6600     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6601   }
6602 }
6603 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
6604 { \@@_error:nn { impossible-delimiter } { left } }
6605 {
6606   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
6607   { \@@_error:nn { impossible-delimiter } { right } }
6608   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
6609 }
6610 \endpgfpicture
6611 }
6612 \group_end:
6613 }

```

`#1` is the left delimiter, `#2` is the right one, `#3` is the subscript and `#4` is the superscript.

```

6614 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
6615 {
6616   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
6617   \dim_set:Nn \l_@@_y_initial_dim
6618   { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
6619   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
6620   \dim_set:Nn \l_@@_y_final_dim
6621   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
6622   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
6623   {
6624     \cs_if_exist:cT
6625     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
6626     {
6627       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }

```

```

6628         \dim_set:Nn \l_@@_y_initial_dim
6629         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6630     }
6631     \cs_if_exist:cT
6632     { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6633     {
6634         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6635         \dim_set:Nn \l_@@_y_final_dim
6636         { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6637     }
6638 }
6639 \dim_set:Nn \l_tmpa_dim
6640 {
6641     \l_@@_y_initial_dim - \l_@@_y_final_dim +
6642     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6643 }
6644 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

6645     \group_begin:
6646     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6647     \tl_if_empty:NF \l_@@_rules_color_tl
6648     { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6649     \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`.

```

6650     \seq_map_inline:Nn \g_@@_cols_vlism_seq
6651     {
6652         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6653         {
6654             \int_compare:nNnT
6655             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6656             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

6657                 \@@_qpoint:n { col - ##1 }
6658                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6659                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6660                 \pgfusepathqstroke
6661             }
6662         }
6663     }

```

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.

```

6664     \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6665     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6666     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6667     {
6668         \bool_lazy_and:nnTF
6669         { \int_compare_p:nNn { ##1 } > 0 }
6670         {
6671             \int_compare_p:nNn
6672             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6673         {
6674             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6675             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6676             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6677             \pgfusepathqstroke
6678         }
6679         { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
6680     }

```


Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:n` or `\clist_map_inline:Nn` is given by curryfication.

```

6681 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6682 { \int_step_inline:n { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6683 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
6684 {
6685   \bool_lazy_and:nnTF
6686   { \int_compare_p:nNn { ##1 } > 0 }
6687   {
6688     \int_compare_p:nNn
6689     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6690   {
6691     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

6692   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
6693   \dim_set:Nn \l_tmpa_dim
6694   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6695   \str_case:nn { #1 }
6696   {
6697     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6698     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
6699     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6700     }
6701   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

6702   \dim_set:Nn \l_tmpb_dim
6703   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6704   \str_case:nn { #2 }
6705   {
6706     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6707     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
6708     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6709   }
6710   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6711   \pgfusepathqstroke
6712   \group_end:
6713 }
6714 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
6715 }

```

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).

```

6716 \str_if_empty:NF \l_@@_submatrix_name_str
6717 {
6718   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
6719   \l_@@_x_initial_dim \l_@@_y_initial_dim
6720   \l_@@_x_final_dim \l_@@_y_final_dim
6721 }
6722 \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`.

```

6723 \begin { pgfscope }
6724 \pgftransformshift
6725 {
6726   \pgfpoint
6727   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6728   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6729 }

```

```

6730 \str_if_empty:NTF \l_@@_submatrix_name_str
6731 { \@@_node_left:nn #1 { } }
6732 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
6733 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

6734 \pgftransformshift
6735 {
6736   \pgfpoint
6737   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6738   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6739 }
6740 \str_if_empty:NTF \l_@@_submatrix_name_str
6741 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
6742 {
6743   \@@_node_right:nnnn #2
6744   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
6745 }
6746 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
6747 \flag_clear_new:n { nicematrix }
6748 \l_@@_code_tl
6749 }

```

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`.

```

6750 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

6751 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
6752 {
6753   \use:e
6754   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
6755 }

```

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 -.

```

6756 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
6757 { #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.

```

6758 \tl_const:Nn \c_@@_integers_alist_tl
6759 {
6760   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
6761   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
6762   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
6763   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
6764 }

```

```

6765 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
6766 {

```

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`.

```

6767 \tl_if_empty:nTF { #2 }
6768 {
6769   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
6770   {
6771     \flag_raise:n { nicematrix }
6772     \int_if_even:nTF { \flag_height:n { nicematrix } }
6773     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
6774     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
6775   }
6776   { #1 }
6777 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

6778 { \c_@@_pgfpointanchor_iii:w { #1 } #2 }
6779 }

```

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. `\c_@@_pgfpointanchor_i:nn`).

```

6780 \cs_new:Npn \c_@@_pgfpointanchor_iii:w #1 #2 -
6781 {
6782   \str_case:nnF { #1 }
6783   {
6784     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
6785     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
6786   }

```

Now the case of a node of the form $i-j$.

```

6787 {
6788   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
6789   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
6790 }
6791 }

```

The command `\c_@@_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`).

```

6792 \cs_new_protected:Npn \c_@@_node_left:nn #1 #2
6793 {
6794   \pgfnode
6795   { rectangle }
6796   { east }
6797   {
6798     \nullfont
6799     \c_math_toggle_token
6800     \tl_if_empty:NF \l_@@_delimiters_color_tl
6801     { \color { \l_@@_delimiters_color_tl } }
6802     \left #1
6803     \vcenter
6804     {
6805       \nullfont
6806       \hrule \@height \l_tmpa_dim
6807       \@depth \c_zero_dim
6808       \@width \c_zero_dim
6809     }
6810     \right .
6811     \c_math_toggle_token
6812   }

```

```

6813 { #2 }
6814 { }
6815 }

```

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.

```

6816 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
6817 {
6818   \pgfnode
6819   { rectangle }
6820   { west }
6821   {
6822     \nullfont
6823     \c_math_toggle_token
6824     \tl_if_empty:NF \l_@@_delimiters_color_tl
6825     { \color { \l_@@_delimiters_color_tl } }
6826     \left .
6827     \vcenter
6828     {
6829       \nullfont
6830       \hrule \@height \l_tmpa_dim
6831       \@depth \c_zero_dim
6832       \@width \c_zero_dim
6833     }
6834     \right #1
6835     \tl_if_empty:NF { #3 } { _ { \smash { #3 } } }
6836     ^ { \smash { #4 } }
6837     \c_math_toggle_token
6838   }
6839   { #2 }
6840   { }
6841 }

```

Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

6842 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
6843 {
6844   \peek_remove_spaces:n
6845   { \@@_brace:nnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
6846 }
6847 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
6848 {
6849   \peek_remove_spaces:n
6850   { \@@_brace:nnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
6851 }
6852 \keys_define:nn { NiceMatrix / Brace }
6853 {
6854   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
6855   left-shorten .default:n = true ,
6856   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
6857   shorten .meta:n = { left-shorten , right-shorten } ,
6858   right-shorten .default:n = true ,
6859   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
6860   yshift .value_required:n = true ,
6861   yshift .initial:n = \c_zero_dim ,
6862   unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
6863 }

```

#1 is the first cell of the rectangle (with the syntax $i-j$; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to `under` or `over`.

```
6864 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
6865 {
6866   \group_begin:
```

The four following token lists correspond to the position of the sub-matrix to which an brace will be attached.

```
6867   \@@_compute_i_j:nn { #1 } { #2 }
6868   \bool_lazy_or:nnTF
6869     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6870     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6871     {
6872       \str_if_eq:nnTF { #5 } { under }
6873       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
6874       { \@@_error:nn { Construct~too~large } { \OverBrace } }
6875     }
6876   {
6877     \keys_set:nn { NiceMatrix / Brace } { #4 }
6878     \pgfpicture
6879     \pgfrememberpicturepositiononpagetrue
6880     \pgf@relevantforpicturesizefalse
6881     \bool_if:NT \l_@@_brace_left_shorten_bool
6882     {
6883       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6884       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
6885       {
6886         \cs_if_exist:cT
6887           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6888           {
6889             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6890             \dim_set:Nn \l_@@_x_initial_dim
6891               { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6892           }
6893       }
6894     }
6895     \bool_lazy_or:nnT
6896       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
6897       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
6898       {
6899         \@@_qpoint:n { col - \l_@@_first_j_tl }
6900         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6901       }
6902     \bool_if:NT \l_@@_brace_right_shorten_bool
6903     {
6904       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
6905       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
6906       {
6907         \cs_if_exist:cT
6908           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6909           {
6910             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6911             \dim_set:Nn \l_@@_x_final_dim
6912               { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6913           }
6914       }
6915     }
6916     \bool_lazy_or:nnT
6917       { \bool_not_p:n \l_@@_brace_right_shorten_bool }
6918       { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
6919       {
6920         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
6921         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6922       }

```

```

6923     \pgfset { inner~sep = \c_zero_dim }
6924     \str_if_eq:nnTF { #5 } { under }
6925     { \@@_underbrace_i:n { #3 } }
6926     { \@@_overbrace_i:n { #3 } }
6927     \endpgfpicture
6928   }
6929   \group_end:
6930 }

```

The argument is the text to put above the brace.

```

6931 \cs_new_protected:Npn \@@_overbrace_i:n #1
6932 {
6933   \@@_qpoint:n { row - \l_@@_first_i_tl }
6934   \pgftransformshift
6935   {
6936     \pgfpoint
6937     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
6938     { \pgf@y + \l_@@_brace_yshift_dim }
6939   }
6940   \pgfnode
6941   { rectangle }
6942   { south }
6943   {
6944     \vbox_top:n
6945     {
6946       \group_begin:
6947       \everycr { }
6948       \halign
6949       {
6950         \hfil ## \hfil \cr
6951         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
6952         \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
6953         \hbox_to_wd:nn
6954         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
6955         { \downbracefill } \cr
6956       }
6957       \group_end:
6958     }
6959   }
6960   { }
6961   { }
6962 }

```

The argument is the text to put under the brace.

```

6963 \cs_new_protected:Npn \@@_underbrace_i:n #1
6964 {
6965   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
6966   \pgftransformshift
6967   {
6968     \pgfpoint
6969     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
6970     { \pgf@y - \l_@@_brace_yshift_dim }
6971   }
6972   \pgfnode
6973   { rectangle }
6974   { north }
6975   {
6976     \group_begin:
6977     \everycr { }
6978     \vbox:n
6979     {
6980       \halign
6981       {

```

```

6982         \hfil ## \hfil \crrc
6983         \hbox_to_wd:nn
6984         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
6985         { \upbracefill } \cr
6986         \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
6987         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
6988     }
6989 }
6990 \group_end:
6991 }
6992 { }
6993 { }
6994 }

```

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.

```

6995 \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.

```

6996 \bool_new:N \c_@@_footnote_bool

6997 \@@_msg_new:nnn { Unknown~key~for~package }
6998 {
6999     The~key~'\l_keys_key_str'~is~unknown. \\
7000     If~you~go~on,~it~will~be~ignored. \\
7001     For~a~list~of~the~available~keys,~type~H~<return>.
7002 }
7003 {
7004     The~available~keys~are~(in~alphabetic~order):~
7005     footnote,~
7006     footnotehyper,~
7007     renew-dots,~and
7008     renew-matrix.
7009 }

```

Maybe we will completely delete the key 'transparent' in a future version.

```

7010 \@@_msg_new:nn { Key~transparent }
7011 {
7012     The~key~'transparent'~is~now~obsolete~(because~it's~name~
7013     is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
7014     and~'renew-matrix'.~However,~you~can~go~on.
7015 }

7016 \keys_define:nn { NiceMatrix / Package }
7017 {
7018     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
7019     renew-dots .value_forbidden:n = true ,
7020     renew-matrix .code:n = \@@_renew_matrix: ,
7021     renew-matrix .value_forbidden:n = true ,
7022     transparent .code:n =
7023     {
7024         \@@_renew_matrix:
7025         \bool_set_true:N \l_@@_renew_dots_bool
7026         \@@_error:n { Key~transparent }

```

```

7027     } ,
7028     transparent .value_forbidden:n = true,
7029     footnote .bool_set:N = \c_@@_footnote_bool ,
7030     footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
7031     unknown .code:n = \@@_error:n { Unknown-key-for-package }
7032   }
7033   \ProcessKeysOptions { NiceMatrix / Package }

7034   \@@_msg_new:nn { footnote-with-footnotehyper-package }
7035   {
7036     You-can't-use-the-option-'footnote'~because-the-package~
7037     footnotehyper~has~already-been-loaded.~
7038     If-you-want,~you-can-use-the-option-'footnotehyper'~and-the-footnotes~
7039     within~the-environments-of~nicematrix~will~be~extracted~with~the~tools~
7040     of~the-package-footnotehyper.\\
7041     If-you-go-on,~the-package-footnote-won't-be-loaded.
7042   }

7043   \@@_msg_new:nn { footnotehyper-with-footnote-package }
7044   {
7045     You-can't-use-the-option-'footnotehyper'~because-the-package~
7046     footnote~has~already-been-loaded.~
7047     If-you-want,~you-can-use-the-option-'footnote'~and-the-footnotes~
7048     within~the-environments-of~nicematrix~will~be~extracted~with~the~tools~
7049     of~the-package-footnote.\\
7050     If-you-go-on,~the-package-footnotehyper~won't-be-loaded.
7051   }

7052   \bool_if:NT \c_@@_footnote_bool
7053   {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7054     \@ifclassloaded { beamer }
7055     { \bool_set_false:N \c_@@_footnote_bool }
7056     {
7057       \@ifpackageloaded { footnotehyper }
7058       { \@@_error:n { footnote-with-footnotehyper-package } }
7059       { \usepackage { footnote } }
7060     }
7061   }

7062   \bool_if:NT \c_@@_footnotehyper_bool
7063   {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7064     \@ifclassloaded { beamer }
7065     { \bool_set_false:N \c_@@_footnote_bool }
7066     {
7067       \@ifpackageloaded { footnote }
7068       { \@@_error:n { footnotehyper-with-footnote-package } }
7069       { \usepackage { footnotehyper } }
7070     }
7071     \bool_set_true:N \c_@@_footnote_bool
7072   }

```

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

The following message will be deleted when we will delete the key `except-corners` for the command `\arraycolor`.

```

7073 \@@_msg_new:nn { key~except-corners }
7074 {
7075   The~key~'except-corners'~has-been-deleted-for~the~command~\token_to_str:N
7076   \arraycolor\ in~the~\token_to_str:N \CodeBefore.~You~should~instead~use~
7077   the~key~'corners'~in~your~\@@_full_name_env:.\
7078   If~you~go~on,~this~key~will~be~ignored.
7079 }
7080 \seq_new:N \c_@@_types_of_matrix_seq
7081 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
7082 {
7083   NiceMatrix ,
7084   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
7085 }
7086 \seq_set_map_x:NNn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
7087 { \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`.

```

7088 \cs_new_protected:Npn \@@_error_too_much_cols:
7089 {
7090   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
7091   {
7092     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
7093     { \@@_fatal:n { too-much-cols-for-matrix } }
7094     {
7095       \bool_if:NF \l_@@_last_col_without_value_bool
7096       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
7097     }
7098   }
7099   { \@@_fatal:n { too-much-cols-for-array } }
7100 }

```

The following command must *not* be protected since it's used in an error message.

```

7101 \cs_new:Npn \@@_message_hdotsfor:
7102 {
7103   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
7104   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
7105 }
7106 \@@_msg_new:nn { negative-weight }
7107 {
7108   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
7109   the~value~'#1'.~If~you~go~on,~the~absolute~value~will~be~used.
7110 }
7111 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
7112 {
7113   You~try~to~use~more~columns~than~allowed~by~your~
7114   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
7115   columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
7116   exterior~columns).~This~error~is~fatal.
7117 }
7118 \@@_msg_new:nn { too-much-cols-for-matrix }
7119 {
7120   You~try~to~use~more~columns~than~allowed~by~your~
7121   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
7122   number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~

```

```

7123 'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
7124 This~error~is~fatal.
7125 }

```

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.

```

7126 \@@_msg_new:nn { too~much~cols~for~array }
7127 {
7128   You~try~to~use~more~columns~than~allowed~by~your~
7129   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
7130   \int_use:N \g_@@_static_num_of_col_int\
7131   ~(plus~the~potential~exterior~ones).~
7132   This~error~is~fatal.
7133 }
7134 \@@_msg_new:nn { last~col~not~used }
7135 {
7136   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
7137   in~your~\@@_full_name_env:~However,~you~can~go~on.
7138 }
7139 \@@_msg_new:nn { columns~not~used }
7140 {
7141   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7142   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
7143   You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
7144 }
7145 \@@_msg_new:nn { in~first~col }
7146 {
7147   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
7148   If~you~go~on,~this~command~will~be~ignored.
7149 }
7150 \@@_msg_new:nn { in~last~col }
7151 {
7152   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
7153   If~you~go~on,~this~command~will~be~ignored.
7154 }
7155 \@@_msg_new:nn { in~first~row }
7156 {
7157   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
7158   If~you~go~on,~this~command~will~be~ignored.
7159 }
7160 \@@_msg_new:nn { in~last~row }
7161 {
7162   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
7163   If~you~go~on,~this~command~will~be~ignored.
7164 }
7165 \@@_msg_new:nn { double~closing~delimiter }
7166 {
7167   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
7168   delimiter.~This~delimiter~will~be~ignored.
7169 }
7170 \@@_msg_new:nn { delimiter~after~opening }
7171 {
7172   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
7173   delimiter.~This~delimiter~will~be~ignored.
7174 }
7175 \@@_msg_new:nn { bad~option~for~line~style }
7176 {
7177   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
7178   is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
7179 }

```

```

7180 \@@_msg_new:nn { Unknown-key-for-xdots }
7181 {
7182   As-for-now,~there-is-only~three-keys~available~here:~'color',~'line-style'~
7183   and~'shorten'~(and-you-try-to-use~'\l_keys_key_str')~.~If-you-go-on,~
7184   this-key-will-be-ignored.
7185 }
7186 \@@_msg_new:nn { Unknown-key-for-rowcolors }
7187 {
7188   As-for-now,~there-is-only~two-keys~available~here:~'cols'~and~'respect-blocks'~
7189   (and-you-try-to-use~'\l_keys_key_str')~.~If-you-go-on,~
7190   this-key-will-be-ignored.
7191 }
7192 \@@_msg_new:nn { ampersand-in-light-syntax }
7193 {
7194   You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
7195   ~you-have-used-the-key~'light-syntax'~.~This-error-is-fatal.
7196 }
7197 \@@_msg_new:nn { Construct-too-large }
7198 {
7199   Your-command~\token_to_str:N #1
7200   can't-be-drawn-because-your-matrix-is-too-small.\\
7201   If-you-go-on,~this-command-will-be-ignored.
7202 }
7203 \@@_msg_new:nn { double-backslash-in-light-syntax }
7204 {
7205   You-can't-use~\token_to_str:N \\~to-separate-rows-because-you-have-used~
7206   the-key~'light-syntax'~.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
7207   (set-by-the-key~'end-of-row')~.~This-error-is-fatal.
7208 }
7209 \@@_msg_new:nn { standard-cline-in-document }
7210 {
7211   The-key~'standard-cline'~is-available-only-in-the-preamble.\\
7212   If-you-go-on-this-command-will-be-ignored.
7213 }
7214 \@@_msg_new:nn { bad-value-for-baseline }
7215 {
7216   The-value-given-to~'baseline'~(\int_use:N \l_tmpa_int)~is-not~
7217   valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
7218   \int_use:N \g_@@_row_total_int\ or-equal-to~'t',~'c'~or~'b'.\\
7219   If-you-go-on,~a-value-of-1~will-be-used.
7220 }
7221 \@@_msg_new:nn { Invalid-name-format }
7222 {
7223   You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
7224   \SubMatrix.\\
7225   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
7226   If-you-go-on,~this-key-will-be-ignored.
7227 }
7228 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
7229 {
7230   You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
7231   \token_to_str:N \SubMatrix\ of-your~\@@_full_name_env:\ but-that~
7232   number-is-not-valid.~If-you-go-on,~it-will-be-ignored.
7233 }
7234 \@@_msg_new:nn { impossible-delimiter }
7235 {
7236   It's-impossible-to-draw-the~#1~delimiter~of-your~
7237   \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
7238   in-that-column.
7239   \bool_if:NT \l_@@_submatrix_slim_bool

```

```

7240     { ~Maybe-you-should-try-without-the-key-'slim'. } \\
7241     If-you-go-on,~this~\token_to_str:N \SubMatrix\ will-be-ignored.
7242 }

7243 \@@_msg_new:nn { width-without-X-columns }
7244 {
7245     You-have-used-the-key-'width'-but-you-have-put-no-'X'-column. \\
7246     If-you-go-on,~that-key-will-be-ignored.
7247 }

7248 \@@_msg_new:nn { empty-environment }
7249 { Your-\@@_full_name_env:\ is-empty.~This-error-is-fatal. }

7250 \@@_msg_new:nn { Delimiter-with-small }
7251 {
7252     You-can't-put-a-delimiter-in-the-preamble-of-your-\@@_full_name_env:\
7253     because-the-key-'small'-is-in-force.\\
7254     This-error-is-fatal.
7255 }

7256 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
7257 {
7258     Your-command~\token_to_str:N\line\{#1\}\{#2\}~in-the-'code-after'~
7259     can't-be-executed-because-a-cell-doesn't-exist.\\
7260     If-you-go-on~this-command-will-be-ignored.
7261 }

7262 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
7263 {
7264     The-name~'#1'~is-already-used-for-a~\token_to_str:N \SubMatrix\
7265     in~this~\@@_full_name_env:.\
7266     If-you-go-on,~this-key-will-be-ignored.\\
7267     For-a-list-of-the-names-already-used,~type-H~<return>.
7268 }
7269 {
7270     The-names-already-defined-in~this~\@@_full_name_env:\ are:~
7271     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
7272 }

7273 \@@_msg_new:nn { r-or-l-with-preamble }
7274 {
7275     You-can't-use-the-key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
7276     You-must-specify-the-alignment-of~your~columns~with~the~preamble~of~
7277     your~\@@_full_name_env:.\
7278     If-you-go-on,~this-key-will-be-ignored.
7279 }

7280 \@@_msg_new:nn { Hdotsfor~in~col-0 }
7281 {
7282     You-can't-use~\token_to_str:N \Hdotsfor\ in-an-exterior-column-of~
7283     the-array.~This-error-is-fatal.
7284 }

7285 \@@_msg_new:nn { bad-corner }
7286 {
7287     #1-is-an-incorrect-specification-for-a-corner~(in-the-keys~
7288     'corners'~and~'except-corners').~The-available~
7289     values-are:~NW,~SW,~NE~and~SE.\\
7290     If-you-go-on,~this-specification-of-corner-will-be-ignored.
7291 }

7292 \@@_msg_new:nn { bad-border }
7293 {
7294     #1-is-an-incorrect-specification-for-a-border~(in-the-key~
7295     'borders'~of~the-command~\token_to_str:N \Block).~The-available~
7296     values-are:~left,~right,~top~and~bottom.\\
7297     If-you-go-on,~this-specification-of-border-will-be-ignored.
7298 }

```

```

7299 \@@_msg_new:nn { tikz~key~without~tikz }
7300 {
7301     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
7302     \Block'~because~you~have~not~loaded~Tikz.~
7303     If~you~go~on,~this~key~will~be~ignored.
7304 }
7305 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
7306 {
7307     In~the~\@@_full_name_env:,~you~must~use~the~key~
7308     'last-col'~without~value.\\
7309     However,~you~can~go~on~for~this~time~
7310     (the~value~'\l_keys_value_tl'~will~be~ignored).
7311 }
7312 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
7313 {
7314     In~\NiceMatrixoptions,~you~must~use~the~key~
7315     'last-col'~without~value.\\
7316     However,~you~can~go~on~for~this~time~
7317     (the~value~'\l_keys_value_tl'~will~be~ignored).
7318 }
7319 \@@_msg_new:nn { Block~too~large-1 }
7320 {
7321     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
7322     too~small~for~that~block. \\
7323 }
7324 \@@_msg_new:nn { Block~too~large-2 }
7325 {
7326     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7327     \g_@@_static_num_of_col_int\
7328     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
7329     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
7330     (&)~at~the~end~of~the~first~row~of~your~
7331     \@@_full_name_env:.\\
7332     If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
7333 }
7334 \@@_msg_new:nn { unknown~column~type }
7335 {
7336     The~column~type~'#1'~in~your~\@@_full_name_env:\
7337     is~unknown. \\
7338     This~error~is~fatal.
7339 }
7340 \@@_msg_new:nn { tabularnote~forbidden }
7341 {
7342     You~can't~use~the~command~\token_to_str:N\tabularnote\
7343     ~in~a~\@@_full_name_env:.~This~command~is~available~only~in~
7344     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
7345     If~you~go~on,~this~command~will~be~ignored.
7346 }
7347 \@@_msg_new:nn { borders~forbidden }
7348 {
7349     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
7350     because~the~option~'rounded-corners'~
7351     is~in~force~with~a~non-zero~value.\\
7352     If~you~go~on,~this~key~will~be~ignored.
7353 }
7354 \@@_msg_new:nn { bottomrule~without~booktabs }
7355 {
7356     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
7357     loaded~'booktabs'.\\
7358     If~you~go~on,~this~key~will~be~ignored.
7359 }

```

```

7360 \@@_msg_new:nn { enumitem~not~loaded }
7361 {
7362   You~can't~use~the~command~\token_to_str:N\tabularnote\
7363   ~because~you~haven't~loaded~'enumitem'.\\
7364   If~you~go~on,~this~command~will~be~ignored.
7365 }
7366 \@@_msg_new:nn { Wrong~last~row }
7367 {
7368   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
7369   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
7370   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
7371   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
7372   without~value~(more~compilations~might~be~necessary).
7373 }
7374 \@@_msg_new:nn { Yet~in~env }
7375 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }
7376 \@@_msg_new:nn { Outside~math~mode }
7377 {
7378   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
7379   (and~not~in~\token_to_str:N \vcenter).\\
7380   This~error~is~fatal.
7381 }
7382 \@@_msg_new:nn { One~letter~allowed }
7383 {
7384   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
7385   If~you~go~on,~it~will~be~ignored.
7386 }
7387 \@@_msg_new:nn { varwidth~not~loaded }
7388 {
7389   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
7390   loaded.\\
7391   If~you~go~on,~your~column~will~behave~like~'p'.
7392 }
7393 \@@_msg_new:nnn { Unknown~key~for~Block }
7394 {
7395   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
7396   \Block.\\ If~you~go~on,~it~will~be~ignored. \\
7397   For~a~list~of~the~available~keys,~type~H<return>.
7398 }
7399 {
7400   The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
7401   hvlines,~l,~line-width,~name,~rounded-corners,~r,~t~and~tikz.
7402 }
7403 \@@_msg_new:nn { Version~of~siunitx~too~old }
7404 {
7405   You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
7406   is~too~old.~You~need~at~least~v~3.0.\\
7407   This~error~is~fatal.
7408 }
7409 \@@_msg_new:nnn { Unknown~key~for~Brace }
7410 {
7411   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
7412   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
7413   If~you~go~on,~it~will~be~ignored. \\
7414   For~a~list~of~the~available~keys,~type~H<return>.
7415 }
7416 {
7417   The~available~keys~are~(in~alphabetic~order):~left-shorten,~
7418   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
7419   right-shorten)~and~yshift.
7420 }

```

```

7421 \@@_msg_new:nnn { Unknown-key-for-CodeAfter }
7422 {
7423   The~key~'\l_keys_key_str'~is-unknown.\\
7424   If~you~go~on,~it~will~be~ignored. \\
7425   For~a~list~of~the~available~keys~in~\token_to_str:N
7426   \CodeAfter,~type~H~<return>.
7427 }
7428 {
7429   The~available~keys~are~(in~alphabetic~order):~
7430   delimiters/color,~
7431   rules~(with~the~subkeys~'color'~and~'width'),~
7432   sub-matrix~(several~subkeys)~
7433   and~xdots~(several~subkeys).~
7434   The~latter~is~for~the~command~\token_to_str:N \line.
7435 }
7436 \@@_msg_new:nnn { Unknown-key-for-SubMatrix }
7437 {
7438   The~key~'\l_keys_key_str'~is-unknown.\\
7439   If~you~go~on,~this~key~will~be~ignored. \\
7440   For~a~list~of~the~available~keys~in~\token_to_str:N
7441   \SubMatrix,~type~H~<return>.
7442 }
7443 {
7444   The~available~keys~are~(in~alphabetic~order):~
7445   'delimiters/color',~
7446   'extra-height',~
7447   'hlines',~
7448   'hvlines',~
7449   'left-xshift',~
7450   'name',~
7451   'right-xshift',~
7452   'rules'~(with~the~subkeys~'color'~and~'width'),~
7453   'slim',~
7454   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
7455   and~'right-xshift').\\
7456 }
7457 \@@_msg_new:nnn { Unknown-key-for-notes }
7458 {
7459   The~key~'\l_keys_key_str'~is-unknown.\\
7460   If~you~go~on,~it~will~be~ignored. \\
7461   For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
7462 }
7463 {
7464   The~available~keys~are~(in~alphabetic~order):~
7465   bottomrule,~
7466   code-after,~
7467   code-before,~
7468   enumitem-keys,~
7469   enumitem-keys-para,~
7470   para,~
7471   label-in-list,~
7472   label-in-tabular~and~
7473   style.
7474 }
7475 \@@_msg_new:nnn { Unknown-key-for-RowStyle }
7476 {
7477   The~key~'\l_keys_key_str'~is-unknown~for~the~command~
7478   \token_to_str:N \RowStyle. \\
7479   If~you~go~on,~it~will~be~ignored. \\
7480   For~a~list~of~the~available~keys,~type~H~<return>.
7481 }
7482 {
7483   The~available~keys~are~(in~alphabetic~order):~

```

```

7484     'bold',~
7485     'cell-space-top-limit',~
7486     'cell-space-bottom-limit',~
7487     'cell-space-limits',~
7488     'color',~
7489     'nb-rows'~and~
7490     'rowcolor'.
7491 }

7492 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
7493 {
7494     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7495     \token_to_str:N \NiceMatrixOptions. \\
7496     If~you~go~on,~it~will~be~ignored. \\
7497     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7498 }
7499 {
7500     The~available~keys~are~(in~alphabetic~order):~
7501     allow-duplicate-names,~
7502     cell-space-bottom-limit,~
7503     cell-space-limits,~
7504     cell-space-top-limit,~
7505     code-for-first-col,~
7506     code-for-first-row,~
7507     code-for-last-col,~
7508     code-for-last-row,~
7509     corners,~
7510     create-extra-nodes,~
7511     create-medium-nodes,~
7512     create-large-nodes,~
7513     delimiters~(several~subkeys),~
7514     end-of-row,~
7515     first-col,~
7516     first-row,~
7517     hlines,~
7518     hvlines,~
7519     last-col,~
7520     last-row,~
7521     left-margin,~
7522     letter-for-dotted-lines,~
7523     light-syntax,~
7524     notes~(several~subkeys),~
7525     nullify-dots,~
7526     renew-dots,~
7527     renew-matrix,~
7528     right-margin,~
7529     rules~(with~the~subkeys~'color'~and~'width'),~
7530     small,~
7531     sub-matrix~(several~subkeys),
7532     vlimes,~
7533     xdots~(several~subkeys).
7534 }

7535 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
7536 {
7537     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7538     \{NiceArray\}. \\
7539     If~you~go~on,~it~will~be~ignored. \\
7540     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7541 }
7542 {
7543     The~available~keys~are~(in~alphabetic~order):~
7544     b,~
7545     baseline,~
7546     c,~

```



```

7547 cell-space-bottom-limit,~
7548 cell-space-limits,~
7549 cell-space-top-limit,~
7550 code-after,~
7551 code-for-first-col,~
7552 code-for-first-row,~
7553 code-for-last-col,~
7554 code-for-last-row,~
7555 colortbl-like,~
7556 columns-width,~
7557 corners,~
7558 create-extra-nodes,~
7559 create-medium-nodes,~
7560 create-large-nodes,~
7561 delimiters/color,~
7562 extra-left-margin,~
7563 extra-right-margin,~
7564 first-col,~
7565 first-row,~
7566 hlines,~
7567 hvlines,~
7568 last-col,~
7569 last-row,~
7570 left-margin,~
7571 light-syntax,~
7572 name,~
7573 notes/bottomrule,~
7574 notes/para,~
7575 nullify-dots,~
7576 renew-dots,~
7577 right-margin,~
7578 rules~(with~the~subkeys~'color'~and~'width'),~
7579 small,~
7580 t,~
7581 tabularnote,~
7582 vlines,~
7583 xdots/color,~
7584 xdots/shorten~and~
7585 xdots/line-style.
7586 }

```

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).

```

7587 \@@_msg_new:nnn { Unknown-key-for-NiceMatrix }
7588 {
7589   The~key~'\l_keys_key_str'~is~unknown~for~the~
7590   \@@_full_name_env:. \\\
7591   If~you~go~on,~it~will~be~ignored. \\\
7592   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7593 }
7594 {
7595   The~available~keys~are~(in~alphabetic~order):~
7596   b,~
7597   baseline,~
7598   c,~
7599   cell-space-bottom-limit,~
7600   cell-space-limits,~
7601   cell-space-top-limit,~
7602   code-after,~
7603   code-for-first-col,~
7604   code-for-first-row,~
7605   code-for-last-col,~
7606   code-for-last-row,~

```

```

7607     colortbl-like,~
7608     columns-width,~
7609     corners,~
7610     create-extra-nodes,~
7611     create-medium-nodes,~
7612     create-large-nodes,~
7613     delimiters~(several~subkeys),~
7614     extra-left-margin,~
7615     extra-right-margin,~
7616     first-col,~
7617     first-row,~
7618     hlines,~
7619     hvlines,~
7620     l,~
7621     last-col,~
7622     last-row,~
7623     left-margin,~
7624     light-syntax,~
7625     name,~
7626     nullify-dots,~
7627     r,~
7628     renew-dots,~
7629     right-margin,~
7630     rules~(with~the~subkeys~'color'~and~'width'),~
7631     small,~
7632     t,~
7633     vlines,~
7634     xdots/color,~
7635     xdots/shorten~and~
7636     xdots/line-style.
7637 }

7638 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
7639 {
7640   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7641   \{NiceTabular\}. \\
7642   If~you~go~on,~it~will~be~ignored. \\
7643   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7644 }
7645 {
7646   The~available~keys~are~(in~alphabetic~order):~
7647   b,~
7648   baseline,~
7649   c,~
7650   cell-space-bottom-limit,~
7651   cell-space-limits,~
7652   cell-space-top-limit,~
7653   code-after,~
7654   code-for-first-col,~
7655   code-for-first-row,~
7656   code-for-last-col,~
7657   code-for-last-row,~
7658   colortbl-like,~
7659   columns-width,~
7660   corners,~
7661   create-extra-nodes,~
7662   create-medium-nodes,~
7663   create-large-nodes,~
7664   extra-left-margin,~
7665   extra-right-margin,~
7666   first-col,~
7667   first-row,~
7668   hlines,~
7669   hvlines,~

```

```

7670 last-col,~
7671 last-row,~
7672 left-margin,~
7673 light-syntax,~
7674 name,~
7675 notes/bottomrule,~
7676 notes/para,~
7677 nullify-dots,~
7678 renew-dots,~
7679 right-margin,~
7680 rules~(with~the~subkeys~'color'~and~'width'),~
7681 t,~
7682 tabularnote,~
7683 vlines,~
7684 xdots/color,~
7685 xdots/shorten~and~
7686 xdots/line-style.
7687 }

7688 \@@_msg_new:nnn { Duplicate-name }
7689 {
7690   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
7691   the~same~environment~name~twice.~You~can~go~on,~but,~
7692   maybe,~you~will~have~incorrect~results~especially~
7693   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
7694   message~again,~use~the~key~'allow-duplicate-names'~in~
7695   '\token_to_str:N \NiceMatrixOptions'.\\
7696   For~a~list~of~the~names~already~used,~type~H~<return>. \\
7697 }
7698 {
7699   The~names~already~defined~in~this~document~are:~
7700   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
7701 }

7702 \@@_msg_new:nn { Option-auto-for-columns-width }
7703 {
7704   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
7705   If~you~go~on,~the~key~will~be~ignored.
7706 }

```

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 StackExchange⁷³, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁷⁴

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`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

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

⁷³cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁷⁴Before 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 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.
Option `allow-duplicate-names`.

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 `pdflatex` 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 StackExchange⁷⁵, 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`.

⁷⁵cf. 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`.

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	
@@ commands:	
<code>\@@_Block:</code>	1221, 5512
<code>\@@_Block_i</code>	5517, 5518, 5522
<code>\@@_Block_ii:nnnnn</code>	5522, 5523
<code>\@@_Block_iv:nnnnn</code>	5560, 5564
<code>\@@_Block_iv:nnnnnn</code>	5779, 5781
<code>\@@_Block_v:nnnnn</code>	5561, 5683
<code>\@@_Block_v:nnnnnn</code>	5808, 5811
<code>\@@_Cdots</code>	1141, 1212, 3967
<code>\g_@@_Cdots_lines_tl</code>	1242, 3159
<code>\@@_CodeAfter:</code>	1225, 6386
<code>\@@_CodeAfter_i:</code>	875, 2806, 2851, 6387
<code>\@@_CodeAfter_ii:n</code> ..	6386, 6387, 6388, 6398
<code>\@@_CodeAfter_iv:n</code>	6391, 6393
<code>\@@_CodeAfter_keys:</code>	3100, 3119
<code>\@@_CodeBefore:w</code>	1384, 1386
<code>\@@_CodeBefore_keys:</code>	1364, 1381
<code>\@@_Ddots</code>	1143, 1214, 3999
<code>\g_@@_Ddots_lines_tl</code>	1245, 3157
<code>\g_@@_HVdotsfor_lines_tl</code>	1247, 3155, 4083, 4160, 7103
<code>\@@_Hdotsfor:</code>	1146, 1219, 4059
<code>\@@_Hdotsfor:nnnn</code>	4085, 4098
<code>\@@_Hdotsfor_i</code>	4068, 4074, 4081
<code>\@@_Hline:</code>	1217, 5061
<code>\@@_Hline_i:n</code>	5061, 5062, 5068
<code>\@@_Hline_ii:nn</code>	5065, 5068
<code>\@@_Hline_iii:n</code>	5066, 5069
<code>\@@_Hspace:</code>	1218, 4053
<code>\@@_Iddots</code>	1144, 1215, 4023
<code>\g_@@_Iddots_lines_tl</code>	1246, 3158
<code>\@@_Ldots</code>	1140, 1145, 1211, 3951
<code>\g_@@_Ldots_lines_tl</code>	1243, 3160
<code>\l_@@_Matrix_bool</code>	238, 1659, 1683, 2412, 2750, 2754, 2762, 2919
<code>\l_@@_NiceArray_bool</code>	235, 394, 1288, 1527, 1599, 1722, 1729, 1741, 2412, 2738, 2750, 2754, 2762, 2895, 4878, 4882, 5026, 5036, 5047, 5051, 6290
<code>\g_@@_NiceMatrixBlock_int</code>	220, 5272, 5277, 5280, 5291
<code>\l_@@_NiceTabular_bool</code> ...	183, 236, 880, 1068, 1363, 1366, 1494, 1632, 1636, 1730, 1742, 2739, 2950, 2971, 2980, 5593, 5692, 6298
<code>\@@_NotEmpty:</code>	1227, 2942
<code>\@@_OnlyMainNiceMatrix:n</code>	1223, 4716
<code>\@@_OnlyMainNiceMatrix_i:n</code>	4719, 4726, 4729
<code>\@@_OverBrace</code>	3094, 6847
<code>\@@_RowStyle:n</code>	1228, 4311
<code>\@@_S:</code>	208, 1772
<code>\@@_SubMatrix</code>	3092, 6537
<code>\@@_SubMatrix_in_code_before</code> ...	1356, 6517
<code>\@@_UnderBrace</code>	3093, 6842
<code>\@@_V:</code>	1687, 1768
<code>\@@_Vdots</code>	1142, 1213, 3983
<code>\g_@@_Vdots_lines_tl</code>	1244, 3156
<code>\@@_Vdotsfor:</code>	1220, 4158
<code>\@@_Vdotsfor:nnnn</code>	4162, 4173
<code>\@@_W:</code>	1686, 1771, 2269
<code>\@@_X</code>	1780, 2966
<code>\l_@@_X_column_bool</code>	240, 2169, 5558
<code>\l_@@_X_columns_aux_bool</code> ..	267, 1552, 2158
<code>\l_@@_X_columns_dim</code>	268, 1553, 1562, 1568, 2161
<code>\@@_actually_color:</code>	1365, 4398
<code>\@@_actually_diagbox:nnnnnn</code>	5571, 5880, 6310, 6327
<code>\@@_actually_draw_Cdots:</code>	3525, 3529
<code>\@@_actually_draw_Ddots:</code>	3675, 3679
<code>\@@_actually_draw_Iddots:</code>	3723, 3727
<code>\@@_actually_draw_Ldots:</code> ..	3486, 3490, 4149
<code>\@@_actually_draw_Vdots:</code> ..	3607, 3611, 4224
<code>\@@_add_to_colors_seq:nn</code>	4384, 4396, 4397, 4421, 4430, 4439, 4448, 4555
<code>\@@_adjust_pos_of_blocks_seq:</code> ..	3076, 3121
<code>\@@_adjust_pos_of_blocks_seq_i:nnnnn</code> ..	3124, 3126
<code>\@@_adjust_size_box:</code>	953, 980, 2019, 2329, 2830, 2875
<code>\@@_adjust_to_submatrix:nn</code>	3370, 3473, 3512, 3593, 3668, 3716
<code>\@@_adjust_to_submatrix:nnnnnn</code> ..	3377, 3379
<code>\@@_after_array:</code>	1668, 2984
<code>\g_@@_after_col_zero_bool</code>	269, 1108, 2807, 4065
<code>\@@_analyze_end:Nn</code>	2560, 2605
<code>\l_@@_argspec_tl</code>	3949, 3950, 3951, 3967, 3983, 3999, 4023, 4079, 4080, 4081, 4156, 4157, 4158, 4234, 4235, 4236, 6535, 6536, 6537
<code>\@@_array:</code>	1066, 2561, 2588
<code>\@@_arraycolor</code>	1353, 4485
<code>\c_@@_arydshln_loaded_bool</code> ...	24, 36, 1798
<code>\l_@@_auto_columns_width_bool</code>	497, 643, 2672, 2676, 5267

\g_@@_aux_found_bool	1259, 1264, 1388, 1515, 1518	\l_@@_code_before_bool	273, 633, 660, 1082, 1279, 1310, 1524,
\g_@@_aux_tl	241,		2619, 2636, 2654, 2685, 2711, 2745, 2782, 3113
\l_@@_bar_at_end_of_pream_bool	1723, 1858, 2742	\g_@@_code_before_tl	1513, 1522, 1525, 3110
\l_@@_baseline_tl	488, 489, 636, 637, 638,	\l_@@_code_before_tl	272, 632, 1309, 1364, 1525
	639, 1075, 1601, 2365, 2377, 2382, 2384,	\l_@@_code_for_first_col_tl	567, 2820
	2389, 2394, 2484, 2485, 2489, 2494, 2496, 2501	\l_@@_code_for_first_row_tl	571, 891, 5977
\@@_begin_of_NiceMatrix:nn	2917, 2938	\l_@@_code_for_last_col_tl	569, 2864
\@@_begin_of_row:	878, 903, 1171, 2808	\l_@@_code_for_last_row_tl	573, 898, 5980
\l_@@_block_auto_columns_width_bool ..	1507, 2677, 5260, 5265, 5275, 5285	\l_@@_code_tl	261, 6512, 6748
\g_@@_block_box_int	313, 1488,	\l_@@_col_max_int	296, 3236, 3247, 3315, 3375, 3392
	5566, 5580, 5582, 5636, 5648, 5660, 5669, 5679	\l_@@_col_min_int	295, 3241, 3304, 3309, 3373, 3390
\l_@@_block_name_str	5771, 5827, 5904, 5907, 5912	\g_@@_col_total_int	225, 997, 1238, 1318,
\@@_block_tikz:nnnnn	5868, 6200		1329, 1401, 1585, 2250, 2251, 2704, 2705,
\g_@@_blocks_dp_dim	231, 961, 964, 965, 5663, 5666		2785, 2789, 2794, 2795, 2854, 2988, 2990,
\g_@@_blocks_ht_dim	230, 967, 970, 971, 5654, 5657		3002, 3560, 3578, 3638, 4219, 4706, 5318,
\g_@@_blocks_seq	284, 1509, 2422, 5673, 5685, 5779		5328, 5362, 5449, 5791, 6015, 6573, 6622, 6870
\g_@@_blocks_wd_dim	229, 955, 958, 959, 5642, 5645	\l_@@_col_width_dim	222, 223, 1938, 2008, 5597, 5600
\c_@@_booktabs_loaded_bool	25, 39, 1160, 2455	\@@_color_index:n	4545, 4566, 4569
\l_@@_borders_clist	302, 5746,	\l_@@_color_int	4509, 4510, 4527, 4528, 4548, 4559
	5841, 6142, 6158, 6160, 6162, 6164, 6183, 6195	\l_@@_color_tl	304, 4542, 4543, 4553, 4556, 5509, 5584, 5586
\@@_brace:nnnnn	6845, 6850, 6864	\g_@@_colors_seq	1360, 4387, 4391, 4392, 4402
\l_@@_brace_left_shorten_bool	6854, 6881, 6896	\l_@@_colors_seq	4504, 4505, 4549, 4568, 4570
\l_@@_brace_right_shorten_bool	6856, 6902, 6917	\@@_colortbl_like:	1148, 1229
\l_@@_brace_yshift_dim	6859, 6938, 6970	\l_@@_colortbl_like_bool	474, 659, 1229, 1711
\@@_cartesian_color:nn	4411, 4423, 4432, 4557	\c_@@_colortbl_loaded_bool	108, 112, 1179
\@@_cartesian_path:	4415, 4656	\l_@@_cols_tl	4414, 4462, 4496, 4506, 4507, 4557, 4604, 4607
\@@_cartesian_path:n	4463, 4598, 4656	\g_@@_cols_vlism_seq	249, 1277, 1706, 1789, 6650
\@@_cell_begin:w	872, 1812,	\@@_columncolor	1354, 4426
	1939, 2010, 2041, 2168, 2278, 2299, 2320, 2928	\@@_columncolor_preamble	1152, 4704
\l_@@_cell_box	879, 927, 929,	\c_@@_columncolor_regex	67, 1714
	935, 941, 944, 948, 957, 958, 963, 964, 969,	\l_@@_columns_width_dim	221, 644, 788, 2673, 2679, 5273, 5279
	970, 981, 982, 983, 984, 986, 989, 993, 995,	\g_@@_com_or_env_str	253, 256
	1014, 1029, 1031, 1038, 1039, 1052, 1162,	\@@_computations_for_large_nodes:	5389, 5402, 5407
	1272, 1274, 1967, 1971, 1975, 1978, 2009,	\@@_computations_for_medium_nodes: ..	5309, 5378, 5388, 5399
	2020, 2171, 2319, 2330, 2809, 2833, 2836,	\@@_compute_a_corner:nnnnnn	5132, 5134, 5136, 5138, 5151
	2838, 2855, 2878, 2882, 5888, 6008, 6045, 6305	\@@_compute_corners:	3075, 5124
\@@_cell_end:	974, 1814,	\@@_compute_i_j:nn	6546, 6570, 6867
	1958, 2015, 2046, 2177, 2280, 2309, 2325, 2928	\@@_construct_preamble:	1285, 1680
\l_@@_cell_space_bottom_limit_dim ..	477, 550, 984, 4357	\l_@@_corners_cells_seq	288, 4601, 4641, 4792, 4798, 4805,
\l_@@_cell_space_top_limit_dim	476, 548, 982, 4346		4947, 4953, 4960, 5126, 5142, 5146, 5147, 5207
\@@_cellcolor ..	1347, 4465, 4477, 4478, 4687	\l_@@_corners_clist	493, 621, 626, 4759, 4914, 5127
\@@_cellcolor_tabular	1150, 4681	\@@_create_blocks_nodes:	1338, 1433
\g_@@_cells_seq	2599, 2600, 2601, 2603	\@@_create_col_nodes:	2564, 2592, 2611
\@@_center_cell_box:	1917, 1962	\@@_create_diag_nodes:	1335, 3023, 3177
\@@_chessboardcolors	1355, 4470	\@@_create_extra_nodes: ..	1431, 2421, 5299
\@@_cline	154, 1210	\@@_create_large_nodes:	5307, 5383
\@@_cline_i:nn	155, 156, 176, 179	\@@_create_medium_and_large_nodes: ..	5304, 5394
\@@_cline_i:w	156, 157		
\@@_cline_ii:w	161, 163		
\@@_cline_iii:w	160, 164, 165		

<code>\@@_create_medium_nodes:</code>	5305, 5373	<code>\l_@@_end_of_row_tl</code>	507, 508, 561, 2584, 2585, 7206
<code>\@@_create_nodes:</code>	5380, 5391, 5401, 5404, 5445		<code>\c_@@_endpgfortikzpicture_tl</code>	51, 55, 3148, 4271
<code>\@@_create_one_block_node:nnnnn</code>	1439, 1442		<code>\c_@@_enumitem_loaded_bool</code>	26, 42, 367, 687, 692, 703, 708
<code>\@@_create_row_node:</code>	1078, 1111, 1161	<code>\@@_env:</code>	215, 219, 912, 918, 1015, 1021, 1035, 1043, 1087, 1093, 1099, 1174, 1319, 1320, 1325, 1326, 1331, 1332, 1344, 1398, 1399, 1404, 1407, 1410, 1413, 1422, 1423, 1428, 1429, 1455, 2620, 2623, 2625, 2641, 2647, 2650, 2659, 2665, 2668, 2690, 2696, 2699, 2716, 2722, 2728, 2752, 2760, 2774, 2785, 2789, 2795, 3028, 3029, 3034, 3035, 3043, 3049, 3087, 3194, 3196, 3203, 3205, 3206, 3211, 3214, 3276, 3344, 3409, 3420, 3433, 3436, 3455, 3458, 3563, 3566, 3581, 3584, 4112, 4130, 4187, 4205, 4256, 4258, 4277, 4280, 5162, 5181, 5199, 5331, 5333, 5341, 5452, 5461, 5479, 5902, 5907, 5908, 5913, 5922, 5926, 5940, 5945, 5957, 5963, 5964, 5967, 5985, 6022, 6416, 6419, 6589, 6591, 6596, 6598, 6625, 6627, 6632, 6634, 6732, 6744, 6887, 6889, 6908, 6910
<code>\@@_cut_on_hyphen:w</code>	333, 346, 4455, 4460, 4523, 4611, 4612, 4634, 4635, 4665, 4666, 6078, 6087, 6118, 6121, 6147, 6150, 6521, 6526, 6552, 6559	<code>\g_@@_env_int</code>	214, 215, 217, 1506, 1516, 1519, 1674, 5488
<code>\g_@@_ddots_int</code>	3055, 3695, 3696	<code>\@@_error:n</code>	12, 370, 395, 520, 530, 583, 713, 771, 781, 787, 796, 804, 822, 829, 837, 838, 839, 845, 850, 851, 852, 866, 868, 869, 870, 1379, 1546, 1580, 1590, 1662, 1999, 2399, 2460, 2506, 4309, 4484, 4499, 5736, 5774, 6140, 6384, 6497, 6508, 6515, 6862, 7026, 7031, 7058, 7068
<code>\@@_def_env:nnn</code>	2901, 2912, 2913, 2914, 2915, 2916	<code>\@@_error:nn</code>	13, 651, 2077, 2123, 2129, 2153, 3954, 3957, 3970, 3973, 3986, 3989, 4003, 4004, 4009, 4010, 4027, 4028, 4033, 4034, 5140, 6145, 6502, 6574, 6604, 6607, 6873, 6874
<code>\@@_define_com:nnn</code>	6274, 6282, 6283, 6284, 6285, 6286	<code>\@@_error:nnn</code>	14, 4261, 6679, 6714
<code>\@@_delimiter:nnn</code>	2074, 2095, 2103, 2116, 2122, 2128, 6401	<code>\@@_error_too_much_cols:</code>	1751, 7088
<code>\l_@@_delimiters_color_tl</code>	510, 719, 1377, 1624, 1625, 1642, 1643, 6380, 6439, 6440, 6467, 6800, 6801, 6824, 6825	<code>\@@_everycr:</code>	1104, 1184, 1187
<code>\l_@@_delimiters_max_width_bool</code>	511, 717, 1647	<code>\@@_everycr_i:</code>	1104, 1105
<code>\g_@@_delta_x_one_dim</code>	3057, 3698, 3708	<code>\l_@@_except_borders_bool</code>	502, 595, 4878, 4882, 5047, 5051
<code>\g_@@_delta_x_two_dim</code>	3059, 3746, 3756	<code>\@@_exec_code_before:</code>	1279, 1358
<code>\g_@@_delta_y_one_dim</code>	3058, 3700, 3708	<code>\@@_expand_clist:N</code>	338, 1158, 1159
<code>\g_@@_delta_y_two_dim</code>	3060, 3748, 3756	<code>\@@_expand_clist:NN</code>	4604, 4605, 4657
<code>\@@_diagbox:nn</code>	1226, 6306	<code>\l_@@_exterior_arraycolsep_bool</code>	490, 784, 1732, 1744, 2741
<code>\@@_dotfill:</code>	6295	<code>\l_@@_extra_left_margin_dim</code>	505, 611, 1301, 2841
<code>\@@_dotfill_i:</code>	6300, 6302	<code>\l_@@_extra_right_margin_dim</code>	506, 612, 1541, 2886, 3641
<code>\@@_dotfill_ii:</code>	6299, 6302, 6303	<code>\@@_extract_brackets</code>	5853, 6050
<code>\@@_dotfill_iii:</code>	6303, 6304	<code>\@@_extract_coords_values:</code>	5470, 5477
<code>\l_@@_dotted_bool</code>	321, 3768, 4814, 4969, 5249, 5256	<code>\@@_fatal:n</code>	15, 245, 1497, 2051, 2055, 2084, 2569, 2573, 2575, 2608, 4070, 7093, 7096, 7099	
<code>\@@_double_int_eval:n</code>	4230, 4244, 4245	<code>\@@_fatal:nn</code>	16, 1803, 2272
<code>\g_@@_dp_ante_last_row_dim</code>	906, 1195	<code>\l_@@_fill_tl</code>	299, 5738, 5851, 5853
<code>\g_@@_dp_last_row_dim</code>	906, 907, 1198, 1199, 1273, 1274, 1618	<code>\l_@@_final_i_int</code>	3064, 3223, 3228, 3231, 3256, 3264, 3268, 3277, 3285, 3365, 3421, 3502, 3575, 3581, 3584, 4103, 4131, 4199, 4209, 4211
<code>\g_@@_dp_row_zero_dim</code>	926, 927, 1189, 1190, 1611, 2478, 2517			
<code>\@@_draw_Cdots:nnn</code>	3510			
<code>\@@_draw_Ddots:nnn</code>	3666			
<code>\@@_draw_Iddots:nnn</code>	3714			
<code>\@@_draw_Ldots:nnn</code>	3471			
<code>\@@_draw_Vdots:nnn</code>	3591			
<code>\@@_draw_blocks:</code>	2422, 5776			
<code>\@@_draw_dotted_lines:</code>	3074, 3144			
<code>\@@_draw_dotted_lines_i:</code>	3147, 3151			
<code>\l_@@_draw_first_bool</code>	311, 4014, 4038, 4049			
<code>\@@_draw_hlines:</code>	3077, 5043			
<code>\@@_draw_line:</code>	3508, 3553, 3664, 3712, 3760, 3762, 4283, 4872, 5041			
<code>\@@_draw_line_ii:nn</code>	4263, 4267			
<code>\@@_draw_line_iii:nn</code>	4270, 4274			
<code>\@@_draw_standard_dotted_line:</code>	3769, 3800			
<code>\@@_draw_standard_dotted_line_i:</code>	3863, 3867			
<code>\l_@@_draw_tl</code>	300, 5740, 5744, 5829, 6060, 6066, 6068, 6070, 6107, 6108			
<code>\@@_draw_unstandard_dotted_line:</code>	3770, 3772			
<code>\@@_draw_unstandard_dotted_line:n</code>	3775, 3778			
<code>\@@_draw_unstandard_dotted_line:nnn</code>	3780, 3785, 3799			
<code>\@@_draw_vlines:</code>	3078, 4874			
<code>\g_@@_empty_cell_bool</code>	281, 988, 998, 2843, 2890, 3965, 3981, 3997, 4021, 4044, 4055			

```

\l_@@_final_j_int .....
    3065, 3224, 3229, 3236, 3241, 3247, 3257,
    3265, 3269, 3278, 3286, 3364, 3366, 3422,
    3455, 3458, 3466, 4124, 4134, 4136, 4178, 4207
\l_@@_final_open_bool .....
    3067, 3230, 3234, 3237, 3244, 3250, 3254,
    3270, 3499, 3534, 3539, 3550, 3614, 3624,
    3629, 3650, 3687, 3735, 3871, 3886, 3917,
    3918, 4101, 4125, 4137, 4176, 4200, 4212, 4253
\@@_find_extremities_of_line:nnnn ...
    ..... 3218, 3476, 3515, 3596, 3671, 3719
\l_@@_first_col_int .....
    ..... 142, 155, 324, 325, 563, 843,
    878, 1329, 1401, 1594, 1724, 2614, 2634,
    3000, 3560, 3578, 4063, 4625, 4718, 5318,
    5328, 5362, 5410, 5449, 6255, 6261, 6267, 6622
\l_@@_first_i_tl ..... 6548,
    6554, 6555, 6585, 6616, 6625, 6627, 6682,
    6689, 6691, 6773, 6784, 6788, 6884, 6905, 6933
\l_@@_first_j_tl .....
    6549, 6557, 6558, 6589, 6591, 6652, 6665,
    6672, 6674, 6774, 6785, 6789, 6887, 6889, 6899
\l_@@_first_row_int .. 322, 323, 564, 847,
    1236, 1323, 1396, 1609, 2396, 2475, 2503,
    2514, 2997, 3430, 3452, 5311, 5325, 5352,
    5409, 5447, 5919, 5937, 6253, 6413, 6586, 7217
\c_@@_footnote_bool .....
    1483, 1678, 6996, 7029, 7052, 7055, 7065, 7071
\c_@@_footnotehyper_bool . 6995, 7030, 7062
\@@_full_name_env: .....
    254, 7077, 7114, 7121, 7129, 7137, 7141,
    7231, 7249, 7252, 7265, 7270, 7275, 7277,
    7307, 7326, 7331, 7336, 7343, 7369, 7378, 7590
\@@_hdottedline: ..... 1216, 5236
\@@_hdottedline:n ..... 5244, 5246
\@@_hdottedline:i: ..... 5239, 5241
\@@_hline:nnnn . 4892, 5058, 5077, 5250, 6132
\@@_hline_i:nnnn ..... 4895, 4898
\@@_hline_ii:nnnn ..... 4923, 4934, 4967
\@@_hline_iii:nnnn ..... 4971, 4973
\@@_hline_iv:nnn ..... 4970, 5014
\l_@@_hlines_clist .. 317, 575, 589, 594,
    623, 1112, 1114, 1118, 1158, 3077, 5056, 5057
\l_@@_hpos_block_str .... 306, 307, 5493,
    5495, 5497, 5499, 5501, 5503, 5538, 5539,
    5541, 5592, 5601, 5614, 5625, 5676, 5699,
    5703, 5715, 5720, 5752, 5754, 5756, 5758,
    5761, 5764, 5989, 6001, 6012, 6016, 6026, 6038
\l_@@_hpos_cell_str ..... 227, 228,
    1812, 1905, 1907, 2011, 2278, 2321, 5537, 5539
\l_@@_hpos_col_str .....
    .... 1861, 1863, 1865, 1867, 1892, 1904,
    1908, 1910, 1918, 1919, 1922, 1927, 1994, 2145
\l_@@_hpos_of_block_cap_bool .....
    .... 308, 5759, 5762, 5765, 5916, 5986, 6023
\g_@@_ht_last_row_dim .....
    ..... 908, 1196, 1197, 1271, 1272, 1617
\g_@@_ht_row_one_dim .. 934, 935, 1193, 1194
\g_@@_ht_row_zero_dim .....
    .... 928, 929, 1191, 1192, 1612, 2477, 2516
\@@_hvlines_block:nnn ..... 5819, 6114
\l_@@_hvlines_block_bool .. 312, 5748, 5815
\@@_i: ..... 5311, 5313,
    5314, 5315, 5316, 5325, 5331, 5333, 5334,
    5335, 5336, 5341, 5342, 5343, 5344, 5352,
    5355, 5357, 5358, 5359, 5411, 5413, 5416,
    5417, 5421, 5422, 5447, 5452, 5454, 5456,
    5460, 5461, 5472, 5479, 5481, 5483, 5487, 5488
\g_@@_iddots_int ..... 3056, 3743, 3744
\l_@@_in_env_bool .... 234, 394, 1497, 1498
\c_@@_in_preamble_bool . 21, 22, 23, 683, 699
\l_@@_initial_i_int ..... 3062,
    3221, 3296, 3299, 3324, 3332, 3336, 3345,
    3353, 3363, 3410, 3495, 3541, 3543, 3557,
    3563, 3566, 4102, 4103, 4113, 4181, 4191, 4193
\l_@@_initial_j_int .....
    ..... 3063, 3222, 3297, 3304,
    3309, 3315, 3325, 3333, 3337, 3346, 3354,
    3364, 3366, 3411, 3433, 3436, 3444, 3631,
    3633, 3638, 4106, 4116, 4118, 4177, 4178, 4189
\l_@@_initial_open_bool .....
    ..... 3066, 3298, 3302, 3305,
    3312, 3318, 3322, 3338, 3492, 3531, 3538,
    3548, 3614, 3621, 3627, 3681, 3729, 3869,
    3916, 4100, 4107, 4119, 4175, 4182, 4194, 4252
\@@_insert_tabularnotes: ..... 2427, 2430
\@@_instruction_of_type:nnn .....
    ..... 1055, 3959, 3975, 3991, 4014, 4038
\c_@@_integers_alist_tl ..... 6758, 6769
\l_@@_inter_dots_dim .....
    ..... 478, 479, 3071, 3874, 3881,
    3892, 3900, 3907, 3912, 3924, 3932, 5029, 5039
\g_@@_internal_code_after_tl 262, 1848,
    2073, 2094, 2102, 2115, 2121, 2127, 2188,
    3096, 3097, 5076, 5243, 5569, 5878, 6308, 6529
\@@_intersect_our_row:nnnnn ..... 4587
\@@_intersect_our_row_p:nnnnn ..... 4537
\@@_j: ..... 5318, 5320,
    5321, 5322, 5323, 5328, 5331, 5333, 5336,
    5338, 5339, 5341, 5344, 5346, 5347, 5362,
    5365, 5367, 5368, 5369, 5424, 5426, 5429,
    5431, 5435, 5436, 5449, 5452, 5453, 5455,
    5460, 5461, 5473, 5479, 5480, 5482, 5487, 5488
\l_@@_key_nb_rows_int .....
    ..... 226, 4303, 4323, 4331, 4338
\l_@@_l_dim .....
    .... 3847, 3848, 3861, 3862, 3874, 3880,
    3891, 3899, 3907, 3912, 3924, 3925, 3932, 3933
\l_@@_large_nodes_bool 501, 602, 5303, 5307
\g_@@_last_col_found_bool ..... 332,
    1241, 1586, 1654, 2703, 2777, 2852, 2987, 6013
\l_@@_last_col_int .....
    ..... 330, 331, 772, 815, 817, 830, 846,
    867, 1262, 1265, 1589, 1736, 2924, 2926,
    2988, 2990, 3601, 3636, 3956, 3972, 4010,
    4034, 5784, 5789, 5790, 5791, 5794, 5822,
    5836, 5848, 5860, 5872, 5884, 5899, 5940,
    5945, 5953, 5969, 6257, 6263, 6269, 7092, 7115
\l_@@_last_col_without_value_bool ...
    ..... 329, 814, 2989, 7095
\l_@@_last_empty_column_int .....
    ..... 5172, 5173, 5186, 5192, 5205
\l_@@_last_empty_row_int .....
    ..... 5154, 5155, 5168, 5189

```


<code>\l_@@_last_i_tl</code>	<code>\l_@@_name_str</code>
.... 6550, 6561, 6562, 6572, 6585, 6619,	... 499, 653, 914, 917, 1017, 1020, 1095,
6632, 6634, 6682, 6689, 6869, 6884, 6905, 6965	1098, 2624, 2625, 2649, 2650, 2667, 2668,
<code>\l_@@_last_j_tl</code>	2698, 2699, 2724, 2727, 2770, 2773, 2791,
..... 6551, 6564, 6565, 6573, 6596,	2794, 3037, 3042, 3048, 3195, 3196, 3207,
6598, 6655, 6665, 6672, 6870, 6908, 6910, 6920	3210, 3213, 5457, 5460, 5484, 5487, 5909, 5912
<code>\l_@@_last_row_int</code>	<code>\g_@@_names_seq</code> 233, 650, 652, 7700
.... 326, 327, 565, 896, 942, 1124, 1256,	<code>\l_@@_nb_cols_int</code>
1260, 1267, 1574, 1578, 1581, 1593, 1615, 6231, 6242, 6250, 6256, 6262, 6268
2586, 2587, 2816, 2817, 2861, 2862, 2992,	<code>\l_@@_nb_rows_int</code> 6230, 6241, 6259
3481, 3520, 3988, 4004, 4028, 4724, 4732,	<code>\@@_newcolumnntype</code> 1131, 1685, 1686, 1687, 2954
5783, 5786, 5787, 5806, 5822, 5836, 5848,	<code>\@@_node_for_cell:</code> 994, 1001, 1369, 2837, 2887
5860, 5871, 5883, 5897, 5968, 5979, 6265, 7368	<code>\@@_node_for_multicolumn:nn</code> 5468, 5475
<code>\l_@@_last_row_without_value_bool</code> ...	<code>\@@_node_left:nn</code> 6731, 6732, 6792
..... 328, 1258, 1576, 2991	<code>\@@_node_position:</code>
<code>\l_@@_left_delim_dim</code>	1325, 1327, 1331, 1333, 1398, 1400, 1408, 1414
..... 1286, 1290, 1295, 2552, 2839	<code>\@@_node_position_i:</code> 1411, 1415
<code>\g_@@_left_delim_tl</code>	<code>\@@_node_right:nnnn</code> 6741, 6743, 6816
1294, 1485, 1626, 1650, 1720, 2058, 2060, 5028	<code>\g_@@_not_empty_cell_bool</code> 271, 992, 999, 2943
<code>\l_@@_left_margin_dim</code>	<code>\@@_not_in_exterior:nnnnn</code> 4579
..... 503, 605, 1300, 2840, 5025, 5440	<code>\@@_not_in_exterior_p:nnnnn</code> 4515
<code>\l_@@_letter_for_dotted_lines_str</code> ...	<code>\l_@@_notes_above_space_dim</code> 494, 495
..... 795, 806, 807, 1784	<code>\l_@@_notes_bottomrule_bool</code>
<code>\l_@@_letter_vlism_tl</code> 248, 582, 1787 671, 833, 861, 2453
<code>\l_@@_light_syntax_bool</code> 487, 559, 1303, 1536	<code>\l_@@_notes_code_after_tl</code> 669, 2462
<code>\@@_light_syntax_i</code> 2577, 2580	<code>\l_@@_notes_code_before_tl</code> 667, 2434
<code>\@@_line</code> 3095, 4236	<code>\@@_notes_label_in_list:n</code> 363, 382, 390, 679
<code>\@@_line_i:nn</code> 4243, 4250	<code>\@@_notes_label_in_tabular:n</code> . 362, 403, 676
<code>\l_@@_line_width_dim</code>	<code>\l_@@_notes_para_bool</code> .. 665, 831, 859, 2438
..... 305, 5750, 6061, 6099, 6110, 6116,	<code>\@@_notes_style:n</code>
6137, 6157, 6172, 6174, 6184, 6185, 6187, 6198 361, 364, 382, 390, 406, 411, 673
<code>\@@_line_with_light_syntax:n</code> ... 2591, 2595	<code>\l_@@_nullify_dots_bool</code>
<code>\@@_line_with_light_syntax_i:n</code> 496, 600, 3963, 3979, 3995, 4019, 4042
..... 2590, 2596, 2597	<code>\l_@@_number_of_notes_int</code> 360, 397, 407, 417
<code>\@@_math_toggle_token:</code> 182, 976,	<code>\@@_old_CT@arc@</code> 1499, 3117
2810, 2827, 2856, 2872, 6353, 6364, 6951, 6987	<code>\@@_old_cdots</code> 1204, 3980
<code>\g_@@_max_cell_width_dim</code>	<code>\@@_old_ddots</code> 1206, 4020
..... 985, 986, 1508, 2678, 5266, 5292	<code>\@@_old_dotfill</code> 6294, 6297, 6305
<code>\c_@@_max_l_dim</code> 3861, 3866	<code>\@@_old_dotfill:</code> 1224
<code>\l_@@_medium_nodes_bool</code> 500, 601, 5301, 5960	<code>\l_@@_old_iRow_int</code> 263, 1252, 3164
<code>\@@_message_hdotsfor:</code> 7101, 7114, 7121, 7129	<code>\@@_old_ialign:</code> 1077, 1200, 3091, 5778
<code>\@@_msg_new:nn</code> 17, 7010, 7034,	<code>\@@_old_iddots</code> 1207, 4043
7043, 7073, 7106, 7111, 7118, 7126, 7134,	<code>\l_@@_old_jCol_int</code> 264, 1254, 3165
7139, 7145, 7150, 7155, 7160, 7165, 7170,	<code>\@@_old_ldots</code> 1203, 3964
7175, 7180, 7186, 7192, 7197, 7203, 7209,	<code>\@@_old_multicolumn</code> 1233, 4058
7214, 7221, 7228, 7234, 7243, 7248, 7250,	<code>\@@_old_pgfpaintanchor</code> 190, 6750, 6754
7256, 7273, 7280, 7285, 7292, 7299, 7305,	<code>\@@_old_pgful@check@rerun</code> 101, 105
7312, 7319, 7324, 7334, 7340, 7347, 7354,	<code>\@@_old_vdots</code> 1205, 3996
7360, 7366, 7374, 7376, 7382, 7387, 7403, 7702	<code>\@@_open_x_final_dim:</code>
<code>\@@_msg_new:nnn</code> 3449, 3501, 3535, 3689, 3738
..... 18, 6997, 7262, 7393, 7409, 7421,	<code>\@@_open_x_initial_dim:</code>
7436, 7457, 7475, 7492, 7535, 7587, 7638, 7688 3427, 3494, 3532, 3684, 3732
<code>\@@_msg_redirect_name:nn</code>	<code>\@@_open_y_final_dim:</code> 3573, 3625, 3737
..... 19, 790, 1666, 5797, 5800	<code>\@@_open_y_initial_dim:</code>
<code>\@@_multicolumn:nnn</code> 1231, 2218 3555, 3622, 3683, 3731
<code>\g_@@_multicolumn_cells_seq</code>	<code>\@@_overbrace_i:n</code> 6926, 6931
..... 291, 1234, 1281, 2233,	<code>\l_@@_parallelize_diags_bool</code>
3013, 3017, 3018, 5336, 5344, 5466, 5924, 5942 491, 492, 597, 3053, 3693, 3741
<code>\g_@@_multicolumn_sizes_seq</code>	<code>\@@_patch_for_revtext:</code> 1462, 1481
.... 292, 1235, 1282, 2235, 3019, 3020, 5467	<code>\@@_patch_m_preamble:n</code>
<code>\g_@@_name_env_str</code> 252, 257, 2227, 2254, 2287, 2292, 2353
258, 1492, 1493, 2607, 2896, 2897, 2905,	<code>\@@_patch_m_preamble_i:n</code>
2906, 2935, 2948, 2967, 2977, 3115, 6278, 7090 2258, 2259, 2260, 2274

```

\@@_patch_m_preamble_ii:nn .....
..... 2261, 2262, 2263, 2284
\@@_patch_m_preamble_iii:n .... 2264, 2289
\@@_patch_m_preamble_iv:nnn .....
..... 2265, 2266, 2267, 2294
\@@_patch_m_preamble_ix:n ..... 2338, 2356
\@@_patch_m_preamble_v:nnnn 2268, 2269, 2314
\@@_patch_m_preamble_x:n .....
..... 2282, 2312, 2333, 2335, 2359
\@@_patch_node_for_cell: ..... 1027, 1369
\@@_patch_node_for_cell:n 1025, 1051, 1054
\@@_patch_preamble:n .....
.... 1708, 1754, 1793, 1801, 1822, 1855,
2062, 2078, 2080, 2096, 2104, 2130, 2190, 2210
\@@_patch_preamble_i:n 1758, 1759, 1760, 1808
\@@_patch_preamble_ii:nn .....
..... 1761, 1762, 1763, 1819
\@@_patch_preamble_iii:n . 1764, 1824, 1832
\@@_patch_preamble_iii_i:n .... 1827, 1829
\@@_patch_preamble_iv:n .....
..... 1765, 1766, 1767, 1877
\@@_patch_preamble_iv_i:n ..... 1880, 1882
\@@_patch_preamble_iv_ii:w 1885, 1886, 1888
\@@_patch_preamble_iv_iii:nn ... 1889, 1890
\@@_patch_preamble_iv_iv:nn .....
..... 1894, 1896, 1997, 2000, 2160
\@@_patch_preamble_iv_v:nnnnnnnn 1900, 1933
\@@_patch_preamble_ix:nn .....
..... 1776, 1777, 1778, 2082
\@@_patch_preamble_ix_i:nnn .... 2086, 2108
\@@_patch_preamble_v:n ... 1768, 1769, 1983
\@@_patch_preamble_v_i:w . 1986, 1987, 1989
\@@_patch_preamble_v_ii:nn .... 1990, 1991
\@@_patch_preamble_vi:nnnn 1770, 1771, 2003
\@@_patch_preamble_vii:n ..... 1772, 2026
\@@_patch_preamble_vii_i:w 2029, 2030, 2032
\@@_patch_preamble_viii:nn .....
..... 1773, 1774, 1775, 2053
\@@_patch_preamble_viii_i:nn .....
..... 2066, 2069, 2071
\@@_patch_preamble_x:n ... 1779, 1780, 2133
\@@_patch_preamble_x_i:w . 2136, 2137, 2139
\@@_patch_preamble_x_ii:n ..... 2140, 2143
\@@_patch_preamble_xi:n .....
... 1817, 1931, 2024, 2049, 2181, 2192, 2216
\@@_patch_preamble_xii:n ..... 1785, 2184
\@@_patch_preamble_xiii:n ..... 2195, 2213
\@@_pgf_rect_node:nnn ..... 450, 1412, 5962
\@@_pgf_rect_node:nnnn .....
.... 425, 1454, 5451, 5478, 5901, 5956, 6718
\c_@@_pgfortikzpicture_tl 50, 54, 3146, 4269
\@@_pgfpntanchor:n ..... 6746, 6751
\@@_pgfpntanchor_i:nn ..... 6754, 6756
\@@_pgfpntanchor_ii:w ..... 6757, 6765
\@@_pgfpntanchor_iii:w ..... 6778, 6780
\@@_picture_position: .....
..... 1320, 1327, 1333, 1400, 1414, 1415
\g_@@_pos_of_blocks_seq .....
285, 1280, 1438, 1510, 2236, 3005, 3009,
3010, 3123, 4513, 4753, 4908, 5219, 5826, 6318
\g_@@_pos_of_stroken_blocks_seq .....
..... 287, 1511, 4757, 4912, 5838

\g_@@_pos_of_xdots_seq .....
..... 286, 1512, 3361, 4755, 4910
\g_@@_post_action_cell_tl .....
..... 874, 978, 1964, 2170, 4344, 4355
\@@_pre_array: ..... 1250, 1311, 1533
\@@_pre_array_i:w ..... 1307, 1533
\@@_pre_array_ii: ..... 1154, 1283
\@@_pre_code_before: ..... 1313, 1390
\c_@@_preamble_first_col_tl .... 1725, 2802
\c_@@_preamble_last_col_tl .... 1737, 2847
\g_@@_preamble_tl .....
..... 1487, 1688, 1692, 1696, 1702,
1716, 1725, 1734, 1737, 1746, 1750, 1791,
1800, 1810, 1821, 1834, 1935, 2005, 2038,
2065, 2093, 2101, 2114, 2120, 2165, 2186,
2199, 2206, 2215, 2226, 2228, 2276, 2286,
2291, 2296, 2316, 2342, 2349, 2358, 2561, 2588
\@@_pred:n ..... 143,
181, 2926, 4793, 4806, 4948, 4961, 6127, 6132
\@@_provide_pgfsyspdfmark: ... 68, 77, 1482
\@@_put_box_in_flow: ..... 1652, 2361, 2554
\@@_put_box_in_flow_bis:nn .... 1649, 2521
\@@_put_box_in_flow_i: ..... 2367, 2369
\@@_qpoint:n ..... 218, 1446, 1448, 1450,
1452, 2372, 2374, 2386, 2402, 2469, 2471,
2487, 2498, 2509, 3183, 3185, 3187, 3189,
3199, 3201, 3444, 3466, 3495, 3502, 3541,
3543, 3557, 3575, 3631, 3633, 4277, 4280,
4624, 4628, 4644, 4646, 4822, 4824, 4826,
4865, 4868, 4870, 4977, 4979, 4981, 5018,
5021, 5031, 5357, 5367, 5893, 5895, 5897,
5899, 5933, 5953, 5982, 6083, 6085, 6092,
6094, 6171, 6173, 6175, 6182, 6186, 6188,
6332, 6334, 6337, 6339, 6406, 6408, 6616,
6619, 6657, 6674, 6691, 6899, 6920, 6933, 6965
\l_@@_radius_dim .....
482, 483, 2187, 3070, 3506, 3507, 3941, 5238
\l_@@_real_left_delim_dim 2523, 2538, 2553
\l_@@_real_right_delim_dim 2524, 2550, 2556
\@@_recreate_cell_nodes: ..... 1336, 1394
\g_@@_recreate_cell_nodes_bool .....
..... 498, 1169, 1336, 1361, 1368, 1373
\@@_rectanglecolor .....
..... 1348, 4318, 4435, 4468, 4488, 4698
\@@_rectanglecolor:nnn ... 4441, 4450, 4453
\@@_renew_NC@rewrite@S: .... 201, 203, 1240
\@@_renew_dots: ..... 1138, 1230
\l_@@_renew_dots_bool .....
..... 598, 780, 1230, 7018, 7025
\@@_renew_matrix: 775, 779, 6210, 7020, 7024
\l_@@_respect_blocks_bool 4494, 4511, 4534
\@@_restore_iRow_jCol: ..... 3116, 3162
\c_@@_revtex_bool ..... 58, 60, 63, 65, 1481
\l_@@_right_delim_dim .....
..... 1287, 1291, 1297, 2555, 2884
\g_@@_right_delim_tl .. 1296, 1486, 1644,
1650, 1721, 2061, 2090, 2091, 2112, 2117, 5038
\l_@@_right_margin_dim .....
..... 504, 607, 1540, 2885, 3640, 5035, 5443
\@@_rotate: ..... 1222, 4229
\g_@@_rotate_bool .....
.. 239, 951, 979, 1955, 2018, 2328, 2829,
2874, 4229, 5592, 5633, 5638, 5698, 5714, 5889

```


\@@_rotate_cell_box:	\l_@@_submatrix_name_str
.... 939, 979, 2018, 2328, 2829, 2874, 5889	6504, 6576, 6716, 6718, 6730, 6732, 6740, 6744
\l_@@_rounded_corners_dim	\g_@@_submatrix_names_seq
303, 5742, 5861, 6075, 6076, 6111, 6139, 6196 289, 3099, 6501, 6505, 7271
\@@_roundedrectanglecolor	\l_@@_submatrix_right_xshift_dim
1349, 4444 316, 6463, 6703, 6737
\l_@@_row_max_int 294, 3231, 3374, 3391	\g_@@_submatrix_seq ... 297, 1278, 3376, 6527
\l_@@_row_min_int 293, 3299, 3372, 3389	\l_@@_submatrix_slim_bool 6469, 6584, 7239
\g_@@_row_of_col_done_bool	\l_@@_submatrix_vlines_clist
..... 270, 1109, 1491, 2633 320, 6473, 6491, 6664, 6666
\g_@@_row_style_tl	\@@_succ:n
..... 274, 886, 1514, 1947, 4336, 4337,	176, 180,
4339, 4342, 4353, 4364, 4369, 4380, 4381, 5587	1087, 1093, 1098, 1099, 1119, 1450, 1452,
\g_@@_row_total_int 224, 1237, 1317,	1851, 2074, 2204, 2234, 2347, 2374, 2716,
1323, 1396, 1592, 2397, 2504, 2992, 2999,	2722, 2727, 2728, 2752, 2760, 2773, 2774,
3430, 3452, 4144, 5311, 5325, 5352, 5447,	2785, 2789, 2794, 2795, 3466, 3543, 3633,
5806, 5919, 5937, 6413, 6572, 6586, 6869, 7218	4583, 4628, 4644, 4789, 4826, 4870, 4883,
\@@_rowcolor	4944, 4981, 5031, 5052, 5077, 5224, 5226,
1350, 4327, 4417	5228, 5230, 5417, 5421, 5431, 5435, 5897,
\@@_rowcolor_tabular	5899, 5953, 6092, 6094, 6206, 6337, 6339, 6408
1151, 4692	\l_@@_suffix_tl
\@@_rowcolors	5379, 5390,
1351, 4572	5400, 5403, 5452, 5460, 5461, 5479, 5487, 5488
\@@_rowcolors_i:nnnnn	\l_@@_tabular_width_dim
4538, 4574 237, 1071, 1073, 1748, 2978
\l_@@_rowcolors_restart_bool ... 4497, 4526	\l_@@_tabularnote_tl 359, 835, 863, 2426, 2435
\@@_rowlistcolors	\g_@@_tabularnotes_seq
1352, 4501, 4573 358, 398, 2441, 2447, 2463
\g_@@_rows_seq . 2583, 2585, 2587, 2589, 2591	\c_@@_tabularx_loaded_bool ... 27, 45, 2965
\l_@@_rows_tl	\@@_test_hline_in_block:nnnnn
..... 4413, 4461, 4540, 4557, 4605, 4630 4909, 4911, 5080
\l_@@_rules_color_tl	\@@_test_hline_in_stroken_block:nnnn .
..... 265, 534, 1531, 1532, 6647, 6648 4913, 5102
\@@_set_CT@arc@:	\@@_test_if_cell_in_a_block:nn
184, 1532, 6648 5158, 5176, 5194, 5214
\@@_set_CT@arc@_i:	\@@_test_if_cell_in_block:nnnnnnn ...
185, 186 5220, 5222
\@@_set_CT@arc@_ii:	\@@_test_if_math_mode: 242, 1496, 2907
185, 188	\@@_test_in_corner_h:
\@@_set_final_coords:	4914, 4942
3400, 3425	\@@_test_in_corner_v:
\@@_set_final_coords_from_anchor:n ..	4759, 4787
... 3416, 3505, 3536, 3617, 3626, 3692, 3740	\@@_test_vline_in_block:nnnnn
\@@_set_initial_coords: 4754, 4756, 5091
3395, 3414	\@@_test_vline_in_stroken_block:nnnn .
\@@_set_initial_coords_from_anchor:n 4758, 5113
... 3405, 3498, 3533, 3616, 3623, 3686, 3734	\l_@@_the_array_box
\@@_set_size:n	1284, 1299, 1558, 1566, 2414, 2415, 2417, 2420
6228, 6243	\c_@@_tikz_loaded_bool
\c_@@_siunitx_loaded_bool ... 191, 195, 200 28, 49, 1339, 3079, 5734
\c_@@_size_seq	\l_@@_tikz_seq
... 1260, 1265, 1315, 1316, 1317, 1318, 2995	301, 5735, 5864, 5873
\l_@@_small_bool	\g_@@_total_X_weight_int
773, 820, 826, 266, 1157, 1545, 1548, 1567, 2157
848, 883, 1164, 2055, 2084, 2811, 2857, 3068	\l_@@_type_of_col_tl
\@@_standard_cline	818,
139, 1209	819, 2936, 2938, 6235, 6236, 6237, 6245, 6250
\@@_standard_cline:w	\c_@@_types_of_matrix_seq
139, 140 7080, 7081, 7086, 7090
\l_@@_standard_cline_bool .. 475, 546, 1208	\@@_underbrace_i:n
\c_@@_standard_tl	6925, 6963
485, 486, 3767	\@@_update_for_first_and_last_row: ..
\g_@@_static_num_of_col_int 922, 987, 1269, 2831, 2876
..... 298, 1661, 1709, 5794, 7130, 7142, 7327	\@@_use_arraybox_with_notes: ... 1606, 2482
\l_@@_stop_loop_bool	\@@_use_arraybox_with_notes_b: . 1603, 2466
3225, 3226,	\@@_use_arraybox_with_notes_c:
3258, 3271, 3280, 3293, 3294, 3326, 3339, 3348 1604, 1635, 2410, 2480, 2519
\@@_store_in_tmpb_tl	\c_@@_varwidth_loaded_bool ... 29, 33, 1996
6053, 6055	\@@_vdottedline:n
\@@_stroke_block:nnn	2189, 5253
5833, 6057	\@@_vline:nnnn . 1850, 4734, 4889, 5257, 6127
\@@_stroke_borders_block:nnn ... 5845, 6135	
\@@_stroke_horizontal:n .. 6163, 6165, 6180	
\@@_stroke_vertical:n 6159, 6161, 6169	
\@@_sub_matrix:nnnnnnn	
6541, 6567	
\@@_sub_matrix_i:nnnn	
6608, 6614	
\l_@@_submatrix_extra_height_dim	
..... 314, 6459, 6642	
\l_@@_submatrix_hlines_clist	
..... 319, 6471, 6489, 6681, 6683	
\l_@@_submatrix_left_xshift_dim	
..... 315, 6461, 6694, 6727	

<code>\@@_vline_i:nnnn</code>	4739, 4743	7357, 7363, 7375, 7379, 7384, 7390, 7396,
<code>\@@_vline_ii:nnnn</code>	4768, 4779, 4812	7406, 7412, 7413, 7423, 7424, 7438, 7439,
<code>\@@_vline_iii:nnnn</code>	4816, 4818	7455, 7459, 7460, 7478, 7479, 7495, 7496,
<code>\@@_vline_iv:nnn</code>	4815, 4861	7538, 7539, 7590, 7591, 7641, 7642, 7695, 7696
<code>\l_@@_vlines_clist</code>	318, 576, 588, 593, 622, 1159, 1694, 1700, 1731, 1743, 2197, 2204, 2340, 2347, 2740, 3078, 4887, 4888	<code>\{</code>
<code>\l_@@_vpos_col_str</code>	1869, 1872, 1874, 1879, 1901, 1917, 1993, 2146	258, 1775, 2075, 2100, 2914, 6286, 6699, 7258, 7344, 7538, 7641
<code>\l_@@_vpos_of_block_tl</code> ..	309, 310, 5505, 5507, 5599, 5613, 5624, 5702, 5719, 5767, 5769	<code>\}</code>
<code>\@@_w:</code>	1685, 1770, 2268	2085, 2914, 6286, 6708, 7258, 7344, 7538, 7641
<code>\l_@@_weight_int</code>	2142, 2147, 2148, 2151, 2154, 2155, 2157, 2161	<code>\ </code>
<code>\l_@@_width_dim</code>	232, 768, 856, 1558, 1566, 2946, 2947, 2968, 2969	2916, 6285
<code>\g_@@_width_first_col_dim</code>	283, 1490, 1597, 2628, 2832, 2833	<code>\l</code> ..
<code>\g_@@_width_last_col_dim</code>	282, 1489, 1656, 2781, 2877, 2878	7076, 7104, 7114, 7121, 7129, 7130, 7141, 7142, 7217, 7218, 7231, 7237, 7241, 7249, 7252, 7264, 7270, 7282, 7326, 7327, 7328, 7336, 7342, 7349, 7362, 7369, 7370, 7378, 7412
<code>\l_@@_width_used_bool</code>	290, 857, 1543	A
<code>\l_@@_x_final_dim</code>	277, 3402, 3451, 3460, 3461, 3464, 3467, 3468, 3619, 3635, 3643, 3647, 3651, 3653, 3658, 3660, 3690, 3699, 3707, 3747, 3755, 3796, 3811, 3820, 3854, 3906, 3922, 4281, 4867, 5032, 5035, 5037, 5039, 6583, 6599, 6600, 6606, 6703, 6720, 6737, 6904, 6911, 6912, 6918, 6921, 6937, 6954, 6969, 6984	<code>\A</code>
<code>\l_@@_x_initial_dim</code>	275, 3397, 3429, 3438, 3439, 3442, 3445, 3446, 3619, 3634, 3635, 3642, 3647, 3651, 3653, 3655, 3658, 3660, 3699, 3707, 3747, 3755, 3793, 3810, 3820, 3854, 3906, 3920, 3922, 3940, 3942, 4278, 4866, 5022, 5025, 5027, 5029, 6582, 6592, 6593, 6603, 6694, 6719, 6727, 6883, 6890, 6891, 6897, 6900, 6937, 6954, 6969, 6984	6499
<code>\l_@@_xdots_color_tl</code>	509, 523, 3485, 3524, 3605, 3606, 3674, 3722, 3776, 4148, 4223, 4240	<code>\aboverulesep</code>
<code>\l_@@_xdots_down_tl</code> ...	527, 3783, 3804, 3839	2457
<code>\l_@@_xdots_line_style_tl</code>	484, 486, 519, 3767, 3776	<code>\addtocounter</code>
<code>\l_@@_xdots_shorten_dim</code>	480, 481, 525, 3072, 3790, 3791, 3880, 3891, 3899	415
<code>\l_@@_xdots_up_tl</code>	528, 3782, 3803, 3829	<code>\alph</code>
<code>\l_@@_y_final_dim</code>	278, 3403, 3503, 3507, 3545, 3549, 3551, 3576, 3586, 3587, 3701, 3704, 3749, 3752, 3796, 3811, 3819, 3856, 3911, 3930, 4282, 4871, 5020, 6409, 6431, 6446, 6620, 6635, 6636, 6641, 6659, 6676, 6720, 6728, 6738	361
<code>\l_@@_y_initial_dim</code>	276, 3398, 3496, 3506, 3544, 3545, 3549, 3551, 3558, 3568, 3569, 3701, 3706, 3749, 3754, 3793, 3810, 3819, 3856, 3911, 3928, 3930, 3940, 3943, 4279, 4869, 5019, 6407, 6431, 6446, 6617, 6628, 6629, 6641, 6658, 6675, 6719, 6728, 6738	<code>\anchor</code>
<code>\</code>	2574, 2596, 6257, 6263, 6269, 6387, 6999, 7000, 7040, 7049, 7077, 7142, 7147, 7152, 7157, 7162, 7200, 7205, 7211, 7218, 7224, 7225, 7240, 7245, 7253, 7259, 7265, 7266, 7277, 7289, 7296, 7308, 7315, 7322, 7331, 7337, 7344, 7351,	3174, 3175
		<code>\array</code>
		1470
		<code>\arraybackslash</code>
		1948, 2172, 2302
		<code>\arraycolor</code>
		1353, 7076
		<code>\arraycolsep</code>
		606, 608, 610, 1070, 1167, 1290, 1291, 1634, 1638, 2415, 2751, 2755, 2765, 5027, 5037
		<code>\arrayrulecolor</code>
		115
		<code>\arrayrulewidth</code>
		147, 152, 172, 536, 913, 1086, 1088, 1094, 1125, 1629, 1640, 1697, 1703, 1792, 1842, 2200, 2207, 2343, 2350, 2474, 2513, 2640, 2642, 2648, 2658, 2660, 2666, 2689, 2691, 2697, 2715, 2717, 2723, 2749, 2753, 2765, 2768, 4626, 4627, 4629, 4645, 4647, 4837, 4838, 4840, 4851, 4857, 4993, 5004, 5010, 5073, 5292, 6061, 6116, 6137, 6431, 6642, 6646
		<code>\arraystretch</code>
		1166, 3559, 3577, 5589, 5695, 5711, 6618, 6621
		<code>\AtBeginDocument</code>
		23, 30, 93, 109, 192, 198, 365, 479, 481, 483, 495, 685, 701, 3142, 3947, 4077, 4154, 4232, 4265, 6533
		<code>\AtBeginEnvironment</code>
		1232
		<code>\AutoNiceMatrix</code>
		6287
		<code>\AutoNiceMatrixWithDelims</code> ...
		6239, 6279, 6291
		B
		<code>\baselineskip</code>
		118, 124, 1976
		<code>\begingroup</code>
		2221
		<code>\bfseries</code>
		4373, 4376
		<code>\bgroup</code>
		1484
		<code>\Block</code>
		1221, 7295, 7302, 7349, 7396
		<code>\BNiceMatrix</code>
		6225
		<code>\bNiceMatrix</code>
		6222
		<code>\Body</code>
		1307
		<code>\boldmath</code>
		4373, 4376
		bool commands:
		<code>\bool_do_until:Nn</code>
		3226, 3294
		<code>\bool_gset_false:N</code>
		951, 998, 999, 1108, 1241, 1361, 1491, 1515, 1693, 2843, 2890, 5089, 5100, 5111, 5122, 5638

163, 165, 187, 189, 364, 1133, 1475, 1476, 2222, 2245, 2958, 3220, 3282, 3350, 4152, 4227, 5061, 5062, 5068, 5069, 5589, 5695, 5711 \cs_set_nopar:Npn 1072, 1166, 1177, 5472, 5473 \cs_set_nopar:Npx 1073 \cs_set_protected:Npn 6276 \cs_set_protected_nopar:Npn 5567, 5876	
D	
\Ddots 1214, 4003, 4004, 4009, 4010 \ddots 1143, 1206 \diagbox 1226, 5567, 5876 dim commands: \dim_add:Nn 5441 \dim_compare:nNnTF .. 118, 124, 955, 961, 967, 1748, 2673, 2882, 2946, 3442, 3464, 3653, 4340, 4351, 5354, 5364, 5931, 5951, 6305 \dim_compare_p:n 5597 \dim_gzero:N 959, 965, 971 \dim_max:nn 5343, 5347 \dim_min:nn 5335, 5339 \dim_ratio:nn 3708, 3756, 3874, 3879, 3890, 3898, 3907, 3912, 3923, 3931 \dim_set:Nn 5316, 5323, 5334, 5338, 5342, 5346, 5358, 5359, 5368, 5369, 5413, 5426 \dim_set_eq:NN 5314, 5321, 5421, 5435 \dim_sub:Nn 5438 \dim_use:N 5355, 5365, 5416, 5417, 5429, 5430, 5453, 5454, 5455, 5456, 5480, 5481, 5482, 5483 \dim_zero_new:N 5313, 5315, 5320, 5322 \c_max_dim 3429, 3442, 3451, 3464, 5314, 5316, 5321, 5323, 5355, 5365, 5918, 5931, 5936, 5951, 6411, 6412, 6582, 6583, 6603, 6606, 6883, 6897, 6904, 6918 \dotfill 1224, 6294 \dots 1145 \doublerulesep 1843, 4840, 4852, 4993, 5005, 5074 \doublerulesepcolor 121 \downbracefill 6955 \draw 3787	
E	
\egroup 1476, 1669 \else 2222 else commands: \else: 244, 4375 \endarray 1474, 1476, 2565, 2593 \endBNiceMatrix 6226 \endbNiceMatrix 6223 \endgroup 2230 \endNiceArray 2953, 2974, 2983 \endNiceArrayWithDelims 2900, 2910 \endpgfscope 3844, 6358 \endpNiceMatrix 6214 \endsavenotes 1678 \endtabular 1476 \endtabularnotes 2448 \endVNiceMatrix 6220 \endvNiceMatrix 6217 \enskip 2065, 2093, 2101, 2114, 2120 \ensuremath 3964, 3980, 3996, 4020, 4043 \everycr 150, 177, 1187, 6947, 6977 \everypar 1941, 1944	
F	
\fi 129, 1690, 2222, 2225, 4380, 5061, 5078 fi commands: \fi: 246, 4377 \five 3169, 3174 flag commands: \flag_clear_new:n 6747 \flag_height:n 6772 \flag_raise:n 6771 \fontdimen 2403 fp commands: \fp_eval:n 3815 \fp_to_dim:n 3850 \futurelet 134	
G	
\globaldefs 1665, 5799 group commands: \group_insert_after:N 6299, 6300, 6302, 6303	
H	
\halign 1201, 6948, 6980 \hbox 1080, 1630, 2418, 2480, 2519, 2638, 2656, 2683, 2687, 2713, 2747 hbox commands: \hbox:n 84, 86, 89 \hbox_gset:Nn 5581 \hbox_overlap_left:n . 1032, 1040, 2617, 2834 \hbox_overlap_right:n 418, 2779, 2880 \hbox_set:Nn 401, 1029, 1294, 1296, 1621, 1971, 2171, 2525, 2540, 5888 \hbox_set:Nw 879, 1299, 2009, 2319, 2809, 2855 \hbox_set_end: 977, 1542, 2017, 2327, 2828, 2873 \hbox_to_wd:nn 443, 468, 6953, 6983 \Hdotsfor 1219, 7104, 7282 \hdotsfor 1146 \hdottedline 1216	
exp commands:	
\exp_after:wN 207, 1532, 1626, 1644, 1708, 2227, 6648 \exp_args:Nnc 5279 \exp_args:Nne 2938 \exp_args:NNV 2585, 3951, 3967, 3983, 3999, 4023, 4081, 4158, 4236, 6537 \exp_args:NNx 1116, 2203, 2346 \exp_args:Nnx 2153 \exp_args:No 3775 \exp_args:NV 1688, 2228, 2561, 2588, 2590, 6070 \exp_args:Nxx 5560, 5561 \exp_last_unbraced:NV 1364, 3100, 5853 \exp_not:N ... 50, 51, 54, 55, 1792, 1836, 1905, 1907, 1912, 1913, 1914, 2995, 3009, 3017, 3019, 3110, 4344, 4355, 4365, 4710, 5146, 5613, 5624, 5702, 5719, 5856, 6249, 6754 \exp_not:n 1063, 1675, 3111, 4091, 4092, 4168, 4687, 4698, 4700, 4711, 5576, 5676, 5688, 5689, 5820, 5834, 5846, 5858, 5885, 6251, 6315, 6316 \expandafter 2957 \ExplSyntaxOff 75, 1677, 5294 \ExplSyntaxOn 72, 1670, 5287 \extrarowheight ... 3559, 5590, 5696, 5712, 6618	

<code>\heavyrulewidth</code>	2458	4483, 4492, 5261, 5491, 5731, 6105, 6193,
<code>\hfil</code>	1771, 2269, 6950, 6982	6233, 6371, 6376, 6457, 6478, 6487, 6852, 7016
<code>\hfill</code>	147, 172	<code>\l_keys_key_str</code>
<code>\Hline</code>	1217, 5064	6999, 7183, 7189, 7275, 7384, 7395, 7411,
<code>\hline</code>	127	7423, 7438, 7459, 7477, 7494, 7537, 7589, 7640
<code>\hrule</code> 131, 147, 172, 1125, 2458, 6445, 6806, 6830		<code>\keys_set:nn</code>
<code>\hsize</code>	1955	542, 544, 558, 799, 802, 809, 1375, 1383,
<code>\hskip</code>	130	1528, 1529, 1893, 1995, 2042, 2150, 2937,
<code>\Hspace</code>	1218	2949, 2970, 2979, 3120, 3484, 3523, 3604,
<code>\hspace</code>	4056	3673, 3721, 4147, 4222, 4239, 4313, 4487,
<code>\hss</code>	1771, 2269	4508, 5274, 5814, 6378, 6382, 6510, 6577, 6877
I		
<code>\ialign</code>	1077, 1177, 1200, 3091, 5778	<code>\keys_set_known:nn</code>
<code>\iddots</code>	1215, 4027, 4028, 4033, 4034 4013, 4037, 5542, 6062, 6117, 6138
<code>\iddots</code>	79, 1144, 1207	<code>\keys_set_known:nnN</code>
if commands:	 2149, 6246
<code>\if_mode_math:</code>	244, 4371	<code>\l_keys_value_tl</code> 7223, 7310, 7317, 7690
<code>\IfBooleanTF</code>	1533	
<code>\ifnum</code>	129, 4336, 5061, 5078	
<code>\ifstandalone</code>	1503	
<code>\ignorespaces</code>	2252, 4382	
int commands:		
<code>\int_case:nnTF</code>	4001, 4007, 4025, 4031	
<code>\int_compare:nNnTF</code> .. 142, 143, 167, 877,		
878, 889, 896, 932, 942, 1122, 1124, 1256,		
1262, 1267, 1545, 1548, 1574, 1578, 1589,		
1593, 1594, 1661, 1966, 2151, 2231, 2250,		
2436, 2475, 2514, 2586, 2614, 2858, 3236,		
3243, 3247, 3249, 3304, 3311, 3315, 3317,		
3481, 3520, 3601, 3636, 3638, 4144, 4219,		
4323, 4576, 4621, 4639, 4675, 4706, 4723,		
4724, 4731, 4732, 4736, 5023, 5033, 5224,		
5226, 5228, 5230, 5585, 5587, 5640, 5652,		
5979, 6011, 6015, 6088, 6090, 6253, 6255,		
6257, 6261, 6263, 6265, 6267, 6269, 6652, 6654		
<code>\int_compare_p:n</code>		
3383, 3384, 3385, 3386, 4591, 4593, 6080, 6081		
<code>\int_do_until:nNnn</code>	4531	
<code>\int_gadd:Nn</code>	2157, 2249	
<code>\int_gincr:N</code>		
876, 905, 1506, 1816, 1930, 2023, 2048,		
2180, 2709, 2734, 2853, 3695, 3743, 5272, 5566		
<code>\int_if_even:nTF</code>	4476, 6772	
<code>\int_incr:N</code>	397, 1826, 4559	
<code>\int_min:nn</code>	3183,	
3185, 3187, 3189, 3199, 3201, 3364, 3391, 3392		
<code>\int_mod:nn</code>	4547	
<code>\int_step_inline:nn</code> ... 1419, 1425, 3025,		
3031, 3039, 3045, 3181, 4472, 4474, 6665, 6682		
<code>\int_step_inline:nnnn</code> 5156, 5174, 5189, 5192		
<code>\c_zero_int</code>	2056, 4389, 4582	
iow commands:		
<code>\iow_now:Nn</code> ... 70, 96, 1670, 1671, 1672, 1677		
<code>\iow_shipout:Nn</code>	5287, 5288, 5294	
<code>\item</code>	2441, 2447	
K		
<code>\kern</code>	89	
keys commands:		
<code>\keys_define:nn</code>	512,	
532, 539, 617, 663, 715, 722, 766, 810, 824,		
841, 854, 1371, 1859, 2141, 4047, 4285,		
	</	

\NiceMatrixLastEnv	216	\pgftransformshift	434, 459, 3191, 3807, 5996, 6018, 6349, 6359, 6427, 6724, 6734, 6934, 6966
\NiceMatrixOptions	808, 7495, 7695	\pgfusepath	3833, 3843, 4407, 4844, 6100
\NiceMatrixoptions	7314	\pgfusepathqfill	3945, 4997
\noalign	118, 124, 129, 152, 1104, 1183, 5061, 5238, 6952, 6986	\pgfusepathqstroke	4859, 5012, 6178, 6191, 6345, 6660, 6677, 6711
\nobreak	387	\phantom	3964, 3980, 3996, 4020, 4043
\nointerlineskip	6952, 6986	\pNiceMatrix	6213
\normalbaselines	1163	prg commands:	
\NotEmpty	1227	\prg_do_nothing:	77, 201, 529, 1183, 3098
\null	2248	\prg_new_conditional:Nnn	4579, 4587
\nulldelimiterspace	2539, 2551, 6426, 6644	\prg_replicate:nn	407, 2704, 2705, 4095, 4849, 5002, 6256, 6259, 6262, 6268
\nullfont	6437, 6444, 6798, 6805, 6822, 6829	\prg_return_false:	4584, 4596
\numexpr	180, 181	\prg_return_true:	4585, 4595
O			
\omit	142, 2616, 2632, 2708, 2733, 6386, 6387	\ProcessKeysOptions	7033
\OnlyMainNiceMatrix	1223, 4715	\ProvideDocumentCommand	79
\OverBrace	3094, 6874, 7412	\ProvidesExplPackage	4
P			
\par	2435, 2443	Q	
\path	6205	\quad	388
peek commands:		quark commands:	
\peek_meaning:NTF	185, 399, 1134, 2959	\q_stop	139, 140, 156, 157, 160, 161, 163, 164, 165, 186, 188, 333, 346, 1364, 1386, 1532, 1708, 1781, 1854, 2088, 2110, 2227, 2270, 2577, 2580, 4230, 4244, 4245, 4455, 4460, 4523, 4611, 4612, 4634, 4635, 4665, 4666, 5470, 5477, 5517, 5518, 5522, 5853, 6055, 6078, 6087, 6118, 6121, 6147, 6150, 6228, 6243, 6521, 6526, 6552, 6559, 6648, 6757, 6765
\peek_meaning_ignore_spaces:NTF	2560, 5064	R	
\peek_meaning_remove_ignore_spaces:NTF	175	\raggedleft	1914, 5604, 6363
\peek_remove_spaces:n	4683, 4694, 5514, 6519, 6539, 6844, 6849	\raggedright	1913, 1955, 5605
\pgfdeclareshape	3167	\rectanglecolor	1348, 3106
\pgfextracty	5982	\refstepcounter	416
\pgfgetlastxy	461	regex commands:	
\pgfpathcircle	3939	\regex_const:Nn	67
\pgfpathlineto	4848, 4854, 5001, 5007, 6177, 6190, 6341, 6659, 6676, 6710	\regex_match:nnTF	6499
\pgfpathmoveto	4847, 4853, 5000, 5006, 6176, 6189, 6336, 6658, 6675, 6701	\regex_replace_all:NnN	1713
\pgfpathrectanglecorners	4649, 4841, 4994, 6096	\relax	180, 181
\pgfpointadd	459	\renewcommand	205
\pgfpointanchor	190, 219, 3407, 3418, 3435, 3457, 3565, 3583, 5333, 5341, 5926, 5944, 5964, 5966, 5983, 6020, 6418, 6591, 6598, 6627, 6634, 6746, 6750, 6889, 6910	\RenewDocumentEnvironment	6212, 6215, 6218, 6221, 6224
\pgfpointdiff	460, 1327, 1333, 1400, 1414, 1415	\RequirePackage	1, 3, 9, 10, 11
\pgfpointlineattime	3809	\right	1644, 2535, 2547, 6450, 6810, 6834
\pgfpointorigin	2623, 2790, 3175	\rotate	1222
\pgfpointscale	459	\roundedrectanglecolor	1349, 5856
\pgfpointshapeborder	4277, 4280	\rowcolor	1151, 1350
\pgfrememberpicturepositiononpagetrue	910, 1005, 1092, 1437, 2622, 2646, 2664, 2695, 2721, 2759, 2788, 3153, 3180, 3764, 4276, 4820, 4863, 4975, 5016, 5376, 5386, 5397, 5891, 6064, 6154, 6331, 6404, 6579, 6879	\rowcolors	1351
\pgfscope	3806, 6348	\rowlistcolors	1352
\pgfset	428, 453, 1006, 5997, 6034, 6347, 6425, 6581, 6923	\RowStyle	1228, 7478
\pgfsetbaseline	1004	S	
\pgfsetcornersarced	4648, 6072	\savedanchor	3169
\pgfsetlinewidth	4857, 5010, 6099, 6157, 6646	\savenotes	1483
\pgfsetrectcap	4858, 5011	scan commands:	
\pgfsetroundcap	6344	\scan_stop:	3101
\pgfsetstrokecolor	6070	\scriptstyle	883, 2811, 2857, 3794, 3795, 3829, 3839
\pgfsyspdfmark	73, 74	seq commands:	
\pgftransformrotate	3813	\seq_clear:N	1706
		\seq_clear_new:N	4504, 5126

<code>\seq_count:N</code>	2587, 4392, 4549	<code>\space</code>	257, 258
<code>\seq_gclear:N</code>	1234, 1235, 1277, 1278, 1280, 1509, 1510, 1511, 1512, 2463, 3099	<code>\stepcounter</code>	405, 410
<code>\seq_gclear_new:N</code>	1281, 1282, 1360, 2583, 2599	str commands:	
<code>\seq_gpop_left:Nn</code>	2589, 2601	<code>\c_backslash_str</code>	257
<code>\seq_gput_left:Nn</code>	652, 2233, 2235, 5826	<code>\c_colon_str</code>	807
<code>\seq_gput_right:Nn</code>	398, 1789, 2236, 3361, 4391, 5673, 5685, 5838, 6318, 6505, 6527	<code>\str_case:nn</code>	1910, 5601, 5989, 6001, 6026, 6038, 6695, 6704
<code>\seq_gset_from_clist:Nn</code>	2995, 3009, 3017, 3019	<code>\str_case:nnTF</code>	1601, 1756, 1922, 2256, 2389, 5129, 6769, 6782
<code>\seq_gset_map_x:Nnn</code>	3123	<code>\str_clear_new:N</code>	6576
<code>\seq_gset_split:Nnn</code>	2585, 2600	<code>\str_gclear:N</code>	3115
<code>\seq_if_empty:NnTF</code>	2422, 3005, 3013, 5142, 5864	<code>\str_gset:Nn</code>	1493, 2897, 2906, 2935, 2948, 2967, 2977, 6278
<code>\seq_if_empty_p:N</code>	4601	<code>\str_if_empty:NnTF</code>	914, 1017, 1095, 1492, 2624, 2649, 2667, 2698, 2724, 2770, 2791, 2896, 2905, 3037, 3195, 3207, 5457, 5484, 5537, 5904, 5909, 6716, 6730, 6740
<code>\seq_if_in:NnTF</code>	650, 4641, 4791, 4797, 4804, 4946, 4952, 4959, 5336, 5344, 5924, 5942, 6501, 7090	<code>\str_if_eq:nnTF</code>	104, 256, 1075, 1784, 1787, 1831, 1854, 1884, 1901, 1904, 1917, 1918, 1919, 1985, 2028, 2058, 2090, 2112, 2135, 2194, 2337, 2572, 2574, 2607, 4568, 6068, 6395, 6872, 6924
<code>\seq_item:Nn</code>	1260, 1265, 1315, 1316, 1317, 1318, 4568, 4570	<code>\str_if_eq_p:nn</code>	518, 1720, 1721, 1797, 4615, 4619, 4669, 4673, 5527, 5532
<code>\seq_map_function:Nn</code>	2591	<code>\str_if_in:NnTF</code>	2377, 2489
<code>\seq_map_indexed_inline:Nn</code>	4387, 4402	<code>\str_new:N</code>	227, 252, 306, 499, 806
<code>\seq_map_inline:Nn</code>	1438, 2441, 2447, 2603, 3376, 4538, 4753, 4755, 4757, 4908, 4910, 4912, 5219, 5779, 6650	<code>\str_range:Nnn</code>	2381, 2493
<code>\seq_mapthread_function:Nnn</code>	5465	<code>\str_set:Nn</code>	228, 307, 649, 795, 1812, 1861, 1863, 1865, 1867, 1869, 1872, 1874, 1879, 1892, 1905, 1907, 1993, 1994, 2011, 2145, 2278, 2321, 5493, 5495, 5497, 5499, 5501, 5503, 5505, 5507, 5538, 5541, 5592, 5699, 5715, 5752, 5754, 5756, 5758, 5761, 5764, 5767, 5769, 6012, 6016, 6504
<code>\seq_new:N</code>	233, 249, 284, 285, 286, 287, 288, 289, 291, 292, 297, 301, 358, 7080	<code>\str_set_eq:NN</code>	653, 807, 5539
<code>\seq_put_right:Nn</code>	5206, 5735	<code>\l_tmpa_str</code>	649, 650, 652, 653
<code>\seq_set_eq:NN</code>	4513	<code>\strut</code>	2441, 2447
<code>\seq_set_filter:Nnn</code>	4514, 4536	<code>\strutbox</code>	3559, 3577, 6618, 6621
<code>\seq_set_from_clist:Nn</code>	5146, 7081	<code>\SubMatrix</code>	1356, 3092, 6530, 6574, 7224, 7231, 7237, 7241, 7264, 7441
<code>\seq_set_map_x:Nnn</code>	7086	sys commands:	
<code>\seq_set_split:Nnn</code>	4505	<code>\sys_if_engine_xetex_p:</code>	1049
<code>\seq_use:Nn</code>	5873	<code>\sys_if_output_dvi_p:</code>	1049
<code>\seq_use:Nnnn</code>	3010, 3018, 3020, 5147, 7271, 7700	T	
<code>\l_tmpa_seq</code>	4514, 4536	<code>\tabcolsep</code>	1069, 1633, 1637
<code>\l_tmpb_seq</code>	4513, 4514, 4536, 4538	<code>\tabskip</code>	1188
<code>\setlist</code>	374, 385, 688, 693, 704, 709	<code>\tabularnote</code>	369, 392, 399, 7342, 7362
siunitx commands:		<code>\tabularnotes</code>	2446
<code>\siunitx_cell_begin:w</code>	1918, 2036, 2043	T _E X and L ^A T _E X 2 _ε commands:	
<code>\siunitx_cell_end:</code>	1919, 2046	<code>\@BTnormal</code>	1161
skip commands:		<code>\@addamp</code>	1464, 2222
<code>\skip_gadd:Nn</code>	2680	<code>\@addamp@LaTeX</code>	1464
<code>\skip_gset:Nn</code>	2671	<code>\@addtopreamble</code>	2229
<code>\skip_gset_eq:NN</code>	2678, 2679	<code>\@array</code>	1471, 1475
<code>\skip_horizontal:N</code>	148, 173, 1300, 1301, 1540, 1541, 1596, 1597, 1633, 1634, 1637, 1638, 1656, 1657, 1697, 1703, 1792, 2187, 2200, 2207, 2343, 2350, 2552, 2553, 2555, 2556, 2627, 2628, 2640, 2642, 2658, 2660, 2682, 2689, 2691, 2710, 2715, 2717, 2735, 2744, 2749, 2751, 2753, 2755, 2781, 2839, 2840, 2841, 2844, 2879, 2884, 2885, 2886	<code>\@array@array</code>	1471
<code>\skip_horizontal:n</code>	419, 1052, 1838	<code>\@arraycr</code>	1468
<code>\skip_vertical:N</code>	152, 1086, 1088, 2432, 2457, 5238	<code>\@arraycr@array</code>	1468
<code>\skip_vertical:n</code>	947, 1629, 1640, 5071, 6952, 6986	<code>\@arstrut</code>	2246
<code>\g_tmpa_skip</code>	2671, 2678, 2679, 2680, 2682, 2710, 2735	<code>\@arstrutbox</code>	907, 908, 947, 1190, 1192, 1194, 1197, 1199, 1943, 1954, 1969, 1975, 2303, 2307
<code>\c_zero_skip</code>	1188	<code>\@classx</code>	1466
<code>\smash</code>	6835, 6836	<code>\@classx@array</code>	1466

\@currentenv	6395	4518, 4821, 4864, 4976, 5017, 5377, 5387,
\@depth	6447, 6807, 6831	5398, 5892, 6065, 6155, 6330, 6405, 6580, 6880
\@empty	2229	\pgfsys@getposition
\@finalstrut	1954	1320, 1325, 1331, 1398, 1406, 1409
\@firststampfalse	2222	\pgfsys@markposition
\@gobblethree	74	1034, 1042, 1087, 1173,
\@halignto	1072, 1073	1319, 2620, 2641, 2659, 2690, 2716, 2752, 2784
\@height	132, 147, 172, 6445, 6806, 6830	\pgfutil@check@rerun
\@ifclassloaded	59, 62, 7054, 7064	101, 102
\@ifnextchar	1239, 1475	\reserved@a
\@ifpackageloaded	32,	134
35, 38, 41, 44, 47, 95, 111, 194, 7057, 7067		\rvtx@ifformat@geq
\@mainaux	70,	65
96, 1670, 1671, 1672, 1677, 5287, 5288, 5294		\set@color
\@mkpreamble	1473, 2228	5585, 5888
\@mkpreamble@array	1473	\tikz@library@external@loaded
\@preamble	2247	1500
\@preamerr	2222	tex commands:
\@sharp	2245	\tex_mkern:D
\@tabarray	1074, 1475	83, 85, 87, 90
\@tabular	1472	\tex_the:D
\@tabular@array	1472	208
\@tempwafalse	1690, 2225	\textfont
\@tempwatrue	1689, 2224	2403
\@temptokena	207, 208, 1688, 1708, 2223, 2227	\textit
\@whilesw	1690, 2225	361
\@width	132, 6448, 6808, 6832	\textsuperscript
\@xargarraycr	1469	362, 363
\@xargarraycr@array	1469	\the
\@xarraycr	1467	180, 181, 1690, 1708, 2225, 2227, 2957
\@xarraycr@array	1467	\thetabularnote
\@xhline	135	364
\array@array	1470	\tikzexternaldisable
\bBigg@	1294, 1296	1502
\c@MaxMatrixCols	2925, 7123	\tikzset
\c@tabularnote	2425, 2436, 2464	1341, 1504, 3081
\col@sep	1069, 1070, 1596,	tl commands:
1657, 2627, 2680, 2744, 2844, 2879, 3446, 3468		\c_empty_tl
\CT@arc	115, 116	4308, 5773
\CT@arc@	114, 119, 133, 146, 171, 187, 189, 1499,	\tl_clear:N
2458, 3117, 4856, 5009, 6069, 6156, 6343, 6649		4773, 4784, 4928, 4939, 6060
\CT@drs	121, 122	\tl_clear_new:N
\CT@drsc@	125, 4831, 4832, 4836, 4986, 4987, 4991	4456, 4457, 4506, 4542,
\CT@everycr	1181	4746, 4901, 6522, 6523, 6548, 6549, 6550, 6551
\CT@row@color	1183	\tl_const:Nn
\endarray@array	1474	50, 51, 54, 55, 485, 2802, 2847, 6758
\if@firststamp	2222	\tl_count:n
\if@tempswa	1690, 2225	2384, 2496
\insert@column	1465	\tl_gclear:N
\insert@column@array	1465	874, 1514, 1521, 1692, 2226, 3097, 3102, 4406
\NC@	1133, 2958	\tl_gclear_new:N
\NC@do	2957	1242, 1243, 1244, 1245, 1246, 1247, 1248, 1513
\NC@find	209	\tl_gput_left:Nn
\NC@find@V	1687	1057, 1725, 4708
\NC@list	1690, 2225, 2957	\tl_gput_right:Nn
\NC@rewrite@S	205	1057, 1550, 1737,
\new@ifnextchar	1239	1791, 1834, 1848, 2073, 2094, 2102, 2115,
\newcol@	1135, 1136, 2960, 2961	2121, 2127, 2188, 2993, 3007, 3015, 3108,
\nicematrix@redefine@check@rerun	96, 99	4083, 4160, 4316, 4325, 4337, 4342, 4353,
\pgf@relevantforpicturesizefalse	1322, 1436, 3154, 3765, 3936, 4401,	4364, 4394, 4685, 4696, 5076, 5144, 5243,
		5569, 5817, 5831, 5843, 5854, 5866, 5878, 6308
		\tl_gset:Nn
		1485, 1486, 1487, 1674,
		1696, 1702, 2060, 2061, 2091, 2117, 3110, 4392
		\tl_if_blank:nTF
		4419, 4422, 4428, 4431, 4437, 4440,
		4446, 4449, 4556, 4749, 4783, 4904, 4938, 5516
		\tl_if_blank_p:n
		4614, 4618, 4668, 4672, 4832, 4987, 5526, 5531
		\tl_if_empty:NnTF
		1112, 1522, 1531, 1624, 1642, 2435,
		3077, 3078, 3104, 4314, 4362, 4553, 4636,
		4637, 4762, 4766, 4777, 4917, 4921, 4932,
		5584, 5829, 5851, 6066, 6439, 6647, 6800, 6824
		\tl_if_empty:nTF
		159, 630, 770, 812, 828, 844, 865,
		1444, 2569, 2596, 3485, 3524, 3605, 3674,
		3722, 4148, 4223, 4240, 6496, 6767, 6835, 7103
		\tl_if_empty_p:N
		1731, 1743, 3803, 3804
		\tl_if_empty_p:n
		2426, 5557
		\tl_if_eq:NnTF
		1114,
		1694, 2197, 2340, 2365, 2484, 4887, 5028,
		5038, 5056, 6553, 6556, 6560, 6563, 6664, 6681
		\tl_if_eq:nnTF
		642, 786, 2088, 2110, 4388

\backslash tl_if_eq_p:NN	3767	token commands:	
\backslash tl_if_exist:NTF	1516	\backslash token_to_str:N	
\backslash tl_if_in:NnTF	4522, 4610, 4633, 4664		7075, 7076, 7104, 7194, 7199, 7205,
\backslash tl_if_in:nnTF	345, 2075, 2085, 2100		7223, 7231, 7237, 7241, 7258, 7264, 7282,
\backslash tl_if_single_token:nTF	581, 794		7295, 7301, 7342, 7349, 7362, 7379, 7395,
\backslash tl_if_single_token_p:n	66		7411, 7412, 7425, 7434, 7440, 7478, 7495, 7695
\backslash tl_map_inline:nn	2570		
\backslash tl_new:N	241,		U
	248, 260, 261, 262, 265, 272, 274, 299,	\backslash UnderBrace	3093, 6873, 7412
	300, 304, 309, 359, 484, 488, 507, 509, 510	\backslash unskip	2451
\backslash tl_put_left:Nn	1161, 1369	\backslash upbracefill	6985
\backslash tl_put_right:Nn	632, 1171, 1269, 1309, 1525	use commands:	
\backslash tl_range:nnn	104	\backslash use:N	1060,
\backslash tl_set:Nn	4461,		1304, 1305, 1519, 1537, 1538, 2920, 2940, 4405
	4462, 4524, 4540, 4543, 4620, 4622, 4638,	\backslash use:n	1898, 4241, 4715,
	4640, 4674, 4676, 4745, 5543, 6089, 6091,		5611, 5622, 5700, 5717, 6126, 6131, 6247, 6753
	6122, 6123, 6151, 6152, 6554, 6557, 6561, 6564	\backslash usepackage	7059, 7069
\backslash tl_set_eq:NN	354, 486,	\backslash usepgfmodule	2
	4458, 4459, 4623, 4763, 4918, 6119, 6120,		V
	6148, 6149, 6524, 6525, 6555, 6558, 6562, 6565	vbox commands:	
\backslash tl_set_rescan:Nnn		\backslash vbox:n	89, 6978
	2584, 3950, 4080, 4157, 4235, 6536	\backslash vbox_set_top:Nn	944
\backslash tl_to_str:n	7087	\backslash vbox_to_ht:nn	439, 466, 2531, 2544
tmpc commands:		\backslash vbox_to_zero:n	946
\backslash l_tmpc_dim	279, 1451, 1458, 3188, 3192,	\backslash vbox_top:n	6944
	4626, 4627, 4650, 4827, 4838, 4843, 4848,	\backslash vcenter	1627, 2529, 6442, 6803, 6827, 7379
	4854, 4982, 4996, 5001, 5007, 5898, 5903,	\backslash Vdots	1213, 3986, 3989
	5958, 6086, 6097, 6174, 6177, 6338, 6341, 6349	\backslash vdots	1142, 1205
\backslash l_tmpc_int	4529, 4530, 4531	\backslash Vdotsfor	1220
\backslash l_tmpc_tl		\backslash vfill	442, 468
	4306, 4314, 4319, 4328, 4456, 4458, 4461,	\backslash vline	133
	4623, 4642, 4746, 4762, 4763, 4766, 4771,	\backslash VNiceMatrix	6219
	4773, 4777, 4782, 4784, 4901, 4917, 4918,	\backslash vNiceMatrix	6216
	4921, 4926, 4928, 4932, 4937, 4939, 6119,	\backslash vrule	131, 1943, 2303, 2307
	6127, 6129, 6148, 6165, 6171, 6522, 6524, 6528	\backslash vskip	130
tmpd commands:		\backslash vtop	1084
\backslash l_tmpd_dim	280,		X
	1453, 1459, 3190, 3193, 4647, 4650, 4839,	\backslash xglobal	892, 899, 1156, 2821, 2865
	4843, 4992, 4996, 5900, 5903, 5936, 5947,		Z
	5951, 5954, 5958, 6095, 6098, 6340, 6341, 6359	\backslash Z	6499
\backslash l_tmpd_tl	4457, 4459, 4462, 6120,		
	6124, 6132, 6149, 6161, 6182, 6523, 6525, 6528		

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
6	The color of the rows and columns	12
6.1	Use of <code>colortbl</code>	12
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	12
6.3	Color tools with the syntax of <code>colortbl</code>	16
7	The command <code>\RowStyle</code>	17
8	The width of the columns	17
8.1	Basic tools	17
8.2	The columns <code>V</code> of <code>varwidth</code>	19
8.3	The columns <code>X</code>	19
9	The exterior rows and columns	20
10	The continuous dotted lines	22
10.1	The option <code>nullify-dots</code>	23
10.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	24
10.3	How to generate the continuous dotted lines transparently	25
10.4	The labels of the dotted lines	25
10.5	Customisation of the dotted lines	25
10.6	The dotted lines and the rules	26
11	The <code>\CodeAfter</code>	27
11.1	The command <code>\line</code> in the <code>\CodeAfter</code>	27
11.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	28
11.3	The commands <code>\OverBrace</code> and <code>\UnderBrace</code> in the <code>\CodeAfter</code>	30
12	The notes in the tabulars	30
12.1	The footnotes	30
12.2	The notes of tabular	31
12.3	Customisation of the tabular notes	32
12.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	34
13	Other features	34
13.1	Use of the column type <code>S</code> of <code>siunitx</code>	34
13.2	Alignment option in <code>{NiceMatrix}</code>	35
13.3	The command <code>\rotate</code>	35
13.4	The option <code>small</code>	35
13.5	The counters <code>iRow</code> and <code>jCol</code>	36
13.6	The option <code>light-syntax</code>	37
13.7	Color of the delimiters	37
13.8	The environment <code>{NiceArrayWithDelims}</code>	37

14	Use of Tikz with nicematrix	37
14.1	The nodes corresponding to the contents of the cells	37
14.1.1	The columns V of varwidth	38
14.2	The medium nodes and the large nodes	39
14.3	The nodes which indicate the position of the rules	40
14.4	The nodes corresponding to the command <code>\SubMatrix</code>	41
15	API for the developpers	42
16	Technical remarks	43
16.1	Definition of new column types	43
16.2	Diagonal lines	43
16.3	The empty cells	44
16.4	The option <code>exterior-arraycolsep</code>	44
16.5	Incompatibilities	45
17	Examples	45
17.1	Utilisation of the key 'tikz' of the command <code>\Block</code>	45
17.2	Notes in the tabulars	46
17.3	Dotted lines	47
17.4	Dotted lines which are no longer dotted	48
17.5	Stacks of matrices	49
17.6	How to highlight cells of a matrix	52
17.7	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	54
18	Implementation	55
19	History	227
	Index	235