

# The package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

June 5, 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 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`.<sup>1</sup>

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}`.

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**.<sup>2</sup>

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 5.15b of `nicematrix`, at the date of 2021/06/05.

<sup>1</sup>The 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>

<sup>2</sup>If 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>{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.

**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.**

## Important

Before the version 5.0, it was mandatory to use, for technical reasons, the letters `L`, `C` et `R` instead of `l`, `c` et `r` in the preambles of the environments of `nicematrix`. If we want to be able to go on using these letters, `nicematrix` must be loaded with the option `define-L-C-R`.

```
\usepackage[define-L-C-R]{nicematrix}
```

# 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`.<sup>3</sup>

```
\NiceMatrixOptions{cell-space-limits = 1pt}
$\begin{pNiceMatrix}
\frac{1}{2} & -\frac{1}{2} \\
\frac{1}{3} & \frac{1}{4}
\end{pNiceMatrix}$
```

---

<sup>3</sup>One 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`: `\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}

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

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

## 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.<sup>4</sup>

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}` 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.

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

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

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.<sup>5</sup>

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

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

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

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

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

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:

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;

<sup>4</sup>The spaces after a command `\Block` are deleted.

<sup>5</sup>This 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 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` 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 pt<sup>6</sup>);
- 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`;
- **New 5.15** the keys `hvlines` draws all the vertical and horizontal rules in the block;
- **New 5.14** 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 `\\`).

**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      & tulipe & marguerite & dahlia \\
violette
& \Block[draw=red,fill={RGB}{204,204,255},rounded-corners]{2-2}
&          {\LARGE De très jolies fleurs}
& & souci \\
pervenche & & & lys \\
arum      & iris  & jacinthe  & muguet
\end{NiceTabular}
```

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

## 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.
- 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).

---

<sup>6</sup>This value is the initial value of the *rounded corners* of Tikz.

```

\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}

```

<b>John</b>	12
	13
<b>Steph</b>	8
	18
<b>Sarah</b>	17
	15
<b>Ashley</b>	20
<b>Henry</b>	14
	15
<b>Madison</b>	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`.<sup>7</sup>
- It's possible to draw one or several borders of the cell with the key `borders`.

```

\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.<sup>8</sup>

### 4.5 A small remark

One should remark that the horizontal centering of the contents of the blocks is correct even when an instruction such as `!\quad` has been used in the preamble of the array in order to increase the space between two columns (this is not the case with `\multicolumn`). In the following example, the header “First group” is correctly centered.

<sup>7</sup>If one simply wishes to color the background of a unique celle, there is no point using the command `\Block`: it's possible to use the command `\cellcolor` (when the key `colortbl-like` is used).

<sup>8</sup>One may consider that the default value of the first mandatory argument of `\Block` is 1-1.

```

\begin{NiceTabular}{@{}c!{\qquad}ccc!{\qquad}ccc@{}}
\toprule
& \Block{1-3}{First group} & & \Block{1-3}{Second group} \\
Rank & 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

## 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.

### 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. 9).

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. 38):

```
\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{2-2}
A&B&C&D \\ \hline
\end{NiceTabular}
```

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

## 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 38.

## 5.3 The tools of `nicematrix` for the rules

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

- the keys `hlines`, `vlines` and `hvlines`;
- 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`<sup>9</sup> presented p. 4;
  - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `Vdots`, etc. (cf. p. 18).
- The corners are created by the key `corners` explained below (see p. 9).

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.

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).

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

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$$

### 5.3.2 The key `hvlines`

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

### 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.<sup>10</sup>

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

<sup>9</sup>And also the command `\multicolumn` also it's recommended to use instead `\Block` in the environments of `nicematrix`.

<sup>10</sup>For 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.

			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			

```
\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 & & & \\
& \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		

```
\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

## 5.4 The command `\diagbox`

```

\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}$

```

$x \backslash y$	$e$	$a$	$b$	$c$
$e$	$e$	$a$	$b$	$c$
$a$	$a$	$e$	$c$	$b$
$b$	$b$	$c$	$e$	$a$
$c$	$c$	$b$	$a$	$e$

---

<sup>11</sup>The author of this document considers that type of construction as graphically poor.

## 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}
```

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

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

```
\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`<sup>12</sup>).

*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 rule<sup>13</sup>. 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.

<sup>12</sup>However, one should remark that the package `arydshln` is not fully compatible with `nicematrix`.

<sup>13</sup>In fact, with `array`, this is true only for `\hline` and “|” but not for `\cline`: cf p. 8

## 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.<sup>14</sup>

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

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

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

New commands are available in that `\CodeBefore`: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors`, `\chessboardcolors` and `arraycolor`.<sup>15</sup>

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.

**New 5.15** These commands don’t color the cells which are in the “corners” if the key `corners` is used. This key has been described p. 9.

- 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

<sup>14</sup>If you use Overleaf, Overleaf will do automatically the right number of compilations.

<sup>15</sup>Remark that, in the `\CodeBefore`, PGF/Tikz nodes of the form “(i-j)” are also available to indicate the position to the potential rules: cf. p. 36.

- 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. 17). 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.

```

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

```

$$\begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}$$

We have used the key `r` which aligns all the columns rightwards (cf. p. 31).

- 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).

```

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

```

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

- 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`<sup>16</sup>. 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`.

<sup>16</sup>The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

- 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.<sup>17</sup>
- With the key `respect-blocks` the “rows” alternately colored may extend over several rows if they have to incorporate blocks (created with the command `\Block`: cf. p. 4).

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

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

```
\begin{NiceTabular}{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 \\\
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

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
  \rowcolors{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

<sup>17</sup>Otherwise, the color of a given row relies only upon the parity of its absolute number.

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{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.<sup>18</sup>

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
\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

<sup>18</sup>Up to now, this key is *not* available in `\NiceMatrixOptions`.

## 7 The width of the columns

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` and `W` of the package `array`.

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

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

In the environments of `nicematrix`, it's also possible to fix the *minimal* width of all the columns of an array 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.<sup>19</sup>

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

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

Without surprise, it's possible to fix the minimal width of the columns of all the matrices 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`<sup>20</sup>. The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

<sup>19</sup>The 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).

<sup>20</sup>At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.



```

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

```

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

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

## 8 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`.

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

```

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

```

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

The dotted lines have been drawn with the tools presented p. 18.

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.<sup>21</sup>
- 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. 33) 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).

<sup>21</sup>The users wishing exterior columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 24).

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

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

#### Remarks

- As shown in the previous example, the horizontal and vertical rules doesn'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 38.
- A specification of color present in `code-for-first-row` also applies to a dotted line draw in this 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. 16) 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. 24.

## 9 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`.<sup>22</sup>

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells<sup>23</sup> 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`.<sup>24</sup>

```
\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}
```

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

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

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

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

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}
```

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

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

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}
```

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

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.<sup>25</sup>

<sup>22</sup>The 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`.

<sup>23</sup>The precise definition of a “non-empty cell” is given below (cf. p. 39).

<sup>24</sup>It's also possible to change the color of all theses dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 22.

<sup>25</sup>In `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. 16

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}
```

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

## 9.1 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.

## 9.2 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).

```


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


```

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`<sup>26</sup> 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}

```

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

### 9.3 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`.<sup>27</sup>

<sup>26</sup>We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

<sup>27</sup>The 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.

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`<sup>22</sup> and `\hdotsfor` are redefined within the environments provided by `nicematrix` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` (“automatic dots” of `amsmath`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

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

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

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

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

## 9.4 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. 24) 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`.

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

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

## 9.5 Customisation of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and by the command `\line` in the `\CodeAfter` which is described p. 24) may be customized by three options (specified between square brackets after the command):

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

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

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

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

### The option `xdots/color`

The option `xdots/color` fixes the color of the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 17.

### 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).<sup>28</sup>

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

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

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

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

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

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

## 9.6 The dotted lines and the rules

The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier `|` in the preamble, by the command `\Hline` and by the keys `hlines`, `vlines` and `hvlines` are not drawn within the blocks).<sup>29</sup>

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

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

<sup>28</sup>The 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.

<sup>29</sup>On the other side, the command `\line` in the `\CodeAfter` (cf. p. 24) does *not* create block.

## 10 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.<sup>30</sup>

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. 33.

Moreover, two special commands are available in the `\CodeAfter`: `\line` and `\SubMatrix`.

### 10.1 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. 22).

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}
```

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

It can also be used to draw a diagonal line not parallel to the other diagonal lines (by default, the dotted lines drawn by `\Ddots` are “parallelized”: cf. p. 38).

```
\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}
```

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

### 10.2 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 `.`;

<sup>30</sup>There is also a key `code-before` described p. 12.



- 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.<sup>31</sup>

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

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

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

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

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

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

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

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

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


---

<sup>31</sup>There 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}]`).

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

```


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


```

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. 37.

**New 5.15** It's also possible to specify some delimiters<sup>32</sup> by placing them in the preamble of the environment (for the environments with a preamble: `{NiceArray}`, `{pNiceArray}`, etc.). This syntax is inspired by the extension `blkarray`.

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

```


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


```

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

## 11 The notes in the tabulars

### 11.1 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`.

<sup>32</sup>Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

## 11.2 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@{}}[first-row,code-for-first-row = \bfseries]
\toprule
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 <sup>a</sup>	Jacques	June 5, 2005
Lefebvre <sup>b</sup>	Mathilde	January 23, 1975
Vanesse	Stephany	October 30, 1994
Dupont	Chantal	January 15, 1998

<sup>a</sup> Achard is an old family of the Poitou.

<sup>b</sup> 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. 28. 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`<sup>a</sup>

Last name	First name	Length of life
Churchill	Wiston	91
Nightingale <sup>b,c</sup>	Florence	90
Schoelcher	Victor	89 <sup>d</sup>
Touchet	Marie	89
Wallis	John	87

Some text before the notes.

<sup>a</sup> It's possible to put a note in the caption.

<sup>b</sup> Considered as the first nurse of history.

<sup>c</sup> Nicknamed “the Lady with the Lamp”.

<sup>d</sup> The label of the note is overlapping.

### 11.3 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: `\textsuperscript{#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: `\textsuperscript{#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. 28).

- 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. 40.

## 11.4 Use of `{NiceTabular}` with `threeparttable`

If you wish to use the environment `{NiceTabular}` or `{NiceTabular*}` 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*}}
\makeatother
```

## 12 Other features

### 12.1 Use of the column type `S` of `siunitx`

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

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

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

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

## 12.2 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).

```


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


```

## 12.3 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  $90^\circ$  in the direct sens.

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

```

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

```

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

## 12.4 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`).

```


$$\begin{bmatrix} 1 & -2 & 3 & 4 & 5 \\ 0 & 3 & 2 & 1 & 2 \\ 0 & 1 & 1 & 2 & 3 \end{bmatrix} \begin{matrix} L_2 \leftarrow 2L_1 - L_2 \\ L_3 \leftarrow L_1 + L_3 \end{matrix}$$


```

$$\begin{bmatrix} 1 & -2 & 3 & 4 & 5 \\ 0 & 3 & 2 & 1 & 2 \\ 0 & 1 & 1 & 2 & 3 \end{bmatrix} \begin{matrix} L_2 \leftarrow 2L_1 - L_2 \\ L_3 \leftarrow L_1 + L_3 \end{matrix}$$

One should note that the environment `{NiceMatrix}` with the option `small` is not composed *exactly* as the environment `{smallmatrix}`. Indeed, all the environments of `nicematrix` are constructed upon

`{array}` (of the package `array`) whereas the environment `{smallmatrix}` is constructed directly with an `\halign` of TeX.

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

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

## 12.5 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 column<sup>33</sup>. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `code-before` (cf. p. 12) and in the `\CodeAfter` (cf. p. 24), `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).

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

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

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  $n \times p$  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, excepted in the cells of the potential exterior rows and columns).

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

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

<sup>33</sup>We 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.



## 12.6 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.

```


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


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


```

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.<sup>34</sup>

## 12.7 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`.

```


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


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


```

## 12.8 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.

```


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


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


```

# 13 Use of Tikz with `nicematrix`

## 13.1 The nodes corresponding to the contents of the cells

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

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

However, it's possible to impose the creation of a node with the command `\NotEmpty`.<sup>35</sup>

<sup>34</sup>The 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.

<sup>35</sup>One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 18) and the computation of the "corners" (cf. p. 9).

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.

```

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

```

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).

```

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

```

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 45).

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

## 13.2 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`.<sup>36</sup>

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.

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

<sup>36</sup>There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

The names of the “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.<sup>37</sup>

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

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`.<sup>38</sup>

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

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

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

**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 (with-out use of `margin` nor `extra-margin`).

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

<sup>37</sup>There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 17).

<sup>38</sup>The 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.

### 13.3 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`.

**New 5.14** 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`.

	tulipe	lys
arum		violette mauve
muguet	dahlia	

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

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

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

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

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

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

### 13.4 The nodes corresponding to the command `\SubMatrix`

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 24.

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}\}`).

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

## 14 API for the developpers

The package `nicematrix` provides two variables which are internal but public<sup>39</sup>:

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

These variables contain the code of what we have called the “code-before” and the “code-after”. 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 `\hatchcell` to hatch the current cell (with an optional argument between brackets for the color). It’s possible to program such command `\hatchcell` as follows, explicitly using the public variable `\g_nicematrix_code_before_tl` (this code requires the Tikz library `patterns`: `\usetikzlibrary{patterns}`).

```
ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_hatch:nnn
{
  \tikz \fill [ pattern = north-west-lines , pattern-color = #3 ]
    ( #1 -| #2 ) rectangle ( \int_eval:n { #1 + 1 } -| \int_eval:n { #2 + 1 } ) ;
}

\NewDocumentCommand \hatchcell { ! 0 { black } }
{
  \tl_gput_right:Nx \g_nicematrix_code_before_tl
    { \__pantigny_hatch:nnn { \arabic { iRow } } { \arabic { jCol } } { #1 } }
}
\ExplSyntaxOff
```

Here is an example of use:

```
\begin{NiceTabular}{ccc}[hvlines]
Tokyo & Paris & London \\
Lima & \hatchcell[blue!30]Oslo & Miami \\
Los Angeles & Madrid & Roma
\end{NiceTabular}
```

Tokyo	Paris	London
Lima	Oslo	Miami
Los Angeles	Madrid	Roma

<sup>39</sup>According 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.

## 15 Technical remarks

### 15.1 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 job<sup>40</sup>:

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

The heavy vertical rule won't extend in the exterior rows.<sup>41</sup>

```
$\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.

### 15.2 Diagonal lines

By default, all the diagonal lines<sup>42</sup> 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:

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

<sup>40</sup>The command `\vrule` is a TeX (and not LaTeX) command.

<sup>41</sup>Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

<sup>42</sup>We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

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]`.

### 15.3 The “empty” cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides. 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`.

### 15.4 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 idea<sup>43</sup>. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`<sup>44</sup>. 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).

### 15.5 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=;}
```

<sup>43</sup>In 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).*

<sup>44</sup>And 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.

Up to now, the package `nicematrix` is not compatible with `aastex63`. If you want to use `nicematrix` with `aastex63`, send me an email and I will try to solve the incompatibilities.

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

## 16 Examples

### 16.1 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 11 p. 26.

Let's consider that we wish to number the notes of a tabular with stars.<sup>45</sup>

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<sup>46</sup>

```
\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{}}[first-row,code-for-first-row = \bfseries]
\toprule
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 \\
\end{NiceTabular}
```

<sup>45</sup>Of course, it's realistic only when there is very few notes in the tabular.

<sup>46</sup>In fact: the value of its argument.



```

Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}

```

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.

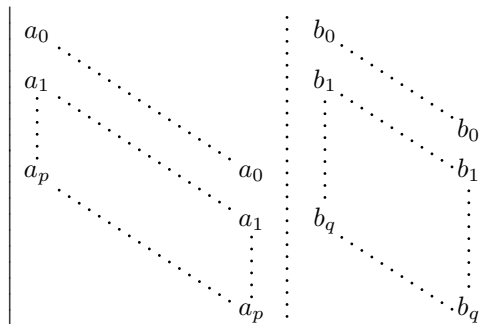
## 16.2 Dotted lines

An example with the resultant of two polynoms:

```

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

```



An example for a linear system:

```

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

```

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

### 16.3 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}}
\[\begin{pNiceMatrix}[last-row,last-col,nullify-dots,xdots/line-style={dashed,blue}]
1&&&\Vdots&&&\Vdots&\backslash
&\Ddots[line-style=standard]&\backslash
&&1&\backslash
&\Cdots[color=blue,line-style=dashed]&&&\blue 0&
&\Cdots&&&\blue 1&&&\Cdots&\blue \leftarrow i&\backslash
&&&&1&\backslash
&&&\Vdots&&\Ddots[line-style=standard]&&&\Vdots&\backslash
&&&&&1&\backslash
&\Cdots&&&\blue 1&\Cdots&&\Cdots&\blue 0&&&\Cdots&\blue \leftarrow j&\backslash
&&&&&&1&\backslash
&&&&&&&\Ddots[line-style=standard]&\backslash
&&&\Vdots&&&&\Vdots&&&1&\backslash
&&&\blue \overset{\uparrow}{i}&&&&\blue \overset{\uparrow}{j}&\backslash
\end{pNiceMatrix}\]
```

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

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

$$\begin{array}{c}
\begin{array}{c} \text{\scriptsize $n$ rows} \end{array}
\begin{array}{c} \left( \begin{array}{cccc} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & \dots & 1 \end{array} \right) \end{array}
\end{array}
\begin{array}{c} \text{\scriptsize $n$ columns} \end{array}$$

## 16.4 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}

```

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

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

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

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

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}

```

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

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

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

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

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}
\[\begin{NiceMatrix}[ r, last-col=6, code-for-last-col = \scriptstyle \color{blue} ]
12 & -8 & 7 & 5 & 3 & \\
3 & -18 & 12 & 1 & 4 & \\
-3 & -46 & 29 & -2 & -15 & \\
9 & 10 & -5 & 4 & 7 & \\[1mm]
12 & -8 & 7 & 5 & 3 & \\
0 & 64 & -41 & 1 & 19 & L_2 \gets L_1-4L_2 \\
0 & -192 & 123 & -3 & -57 & L_3 \gets L_1+4L_3 \\
0 & -64 & 41 & -1 & -19 & L_4 \gets 3L_1-4L_4 \\[1mm]
12 & -8 & 7 & 5 & 3 & \\
0 & 64 & -41 & 1 & 19 & \\
0 & 0 & 0 & 0 & 0 & L_3 \gets 3L_2+L_3 \\[1mm]
12 & -8 & 7 & 5 & 3 & \\
0 & 64 & -41 & 1 & 19 & \\
\CodeAfter [sub-matrix/vlines=4]
\SubMatrix({1-1}{4-5})
\SubMatrix({5-1}{8-5})
\SubMatrix({9-1}{11-5})
\SubMatrix({12-1}{13-5})
\end{NiceMatrix}\]

```

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

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

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

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

## 16.5 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 block<sup>47</sup>).

```

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

```

<sup>47</sup>We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

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

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 command `\hline`, the specifier “|” and the options `hlines`, `vlines` and `hvlines` spread the cells.<sup>48</sup>

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. We create a rectangular Tikz node which encompasses the nodes of the second row with the Tikz library `fit`. This Tikz node is filled after the construction of the matrix. In order to see the text *under* this node, we have to use transparency with the `blend mode` equal to `multiply`.

**Caution** : Some PDF readers are not able to show transparency.<sup>49</sup>

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}
```

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

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

<sup>48</sup>For the command `\cline`, see the remark p. 8.

<sup>49</sup>In Overleaf, the “built-in” PDF viewer does not show transparency. You can switch to the “native” viewer in that case.

$$\begin{bmatrix} 0 \dots\dots\dots 0 \\ 1 \dots\dots\dots 1 \\ 0 \dots\dots\dots 0 \end{bmatrix}$$

We recall that, for a rectangle of merged cells (with the command `\Block`), a Tikz node is created for the set of merged cells with the name *i-j-block* where *i* and *j* are the number of the row and the number of the column of the upper left cell (where the command `\Block` has been issued). If the user has required the creation of the `medium` nodes, a node of this type is also created with a name suffixed by `-medium`.

```

 $\begin{pNiceMatrix}[margin,create-medium-nodes]
  \Block{3-3}<\Large>\{A\} & & 0 \\
  & \hspace*{1cm} & \Vdots \\
  & & 0 \\
  0 & \Cdots & 0 & 0
\CodeAfter
  \tikz \node [highlight = (1-1-block-medium)] {};
\end{pNiceMatrix}$ 

```

$$\left( \begin{array}{c|c} \text{A} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \dots\dots\dots 0 & 0 \end{array} \right)$$

Consider now the following matrix which we have named `example`.

```

 $\begin{pNiceArray}\{ccc\}[name=example,last-col,create-medium-nodes]
a & a + b & a + b + c & L_1 \\
a & a & a + b & L_2 \\
a & a & a & L_3
\end{pNiceArray}$ 

```

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

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

```

\tikzset{mes-options/.style={remember picture,
                             overlay,
                             name prefix = exemple-,
                             highlight/.style = {fill = red!15,
                                                  blend mode = multiply,
                                                  inner sep = 0pt,
                                                  fit = #1}}}

\begin{tikzpicture}[mes-options]
\node [highlight = (1-1) (1-3)] {};
\node [highlight = (2-1) (2-3)] {};
\node [highlight = (3-1) (3-3)] {};
\end{tikzpicture}

```

We obtain the following matrix.

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

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

```

\begin{tikzpicture}[mes-options, name suffix = -medium]
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}

```

We obtain the following matrix.

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

## 16.6 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 `code-before`.

You will find the LaTeX code of that figure in the source file of this document.

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

## 17 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 `expl3`:

```

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.

```

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_@@_tikz_loaded_bool
28 \AtBeginDocument
29   {
30     \ifpackageloaded { arydshln }
31       { \bool_set_true:N \c_@@_arydshln_loaded_bool }
32     { }
33     \ifpackageloaded { booktabs }
34       { \bool_set_true:N \c_@@_booktabs_loaded_bool }
35     { }
36     \ifpackageloaded { enumitem }
37       { \bool_set_true:N \c_@@_enumitem_loaded_bool }
38     { }
39     \ifpackageloaded { tikz }
40     {

```

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.

```

41     \bool_set_true:N \c_@@_tikz_loaded_bool
42     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }

```

```

43     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
44   }
45   {
46     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
47     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
48   }
49 }

```

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`).

```

50 \bool_new:N \c_@@_revtex_bool
51 \ifclassloaded { revtex4-1 }
52   { \bool_set_true:N \c_@@_revtex_bool }
53   { }
54 \ifclassloaded { revtex4-2 }
55   { \bool_set_true:N \c_@@_revtex_bool }
56   { }

```

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

```

57 \cs_if_exist:NT \rvtx@ifformat@geq { \bool_set_true:N \c_@@_revtex_bool }
58 \cs_generate_variant:Nn \tl_if_single_token_p:n { V }

```

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.

```

59 \ProvideDocumentCommand \iddots { }
60   {
61     \mathinner
62     {
63       \tex_mkern:D 1 mu
64       \box_move_up:nn { 1 pt } { \hbox:n { . } }
65       \tex_mkern:D 2 mu
66       \box_move_up:nn { 4 pt } { \hbox:n { . } }
67       \tex_mkern:D 2 mu
68       \box_move_up:nn { 7 pt }
69       { \vbox:n { \kern 7 pt \hbox:n { . } } }
70       \tex_mkern:D 1 mu
71     }
72   }

```

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.

```

73 \AtBeginDocument
74   {
75     \ifpackageloaded { booktabs }
76       { \iow_now:Nn \mainaux \nicematrix@redefine@check@rerun }
77       { }
78     }
79 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
80   {
81     \cs_set_eq:NN \@@_old_pgfulil@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`).

```

82     \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
83     {
84       \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
85       { \@@_old_pgfulil@check@rerun { ##1 } { ##2 } }
86     }

```

```
87 }
```

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

```
88 \bool_new:N \c_@@_colortbl_loaded_bool
89 \AtBeginDocument
90 {
91   \@ifpackageloaded { colortbl }
92     { \bool_set_true:N \c_@@_colortbl_loaded_bool }
93 }
```

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.

```
94   \cs_set_protected:Npn \CT@arc@ { }
95   \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
96   \cs_set:Npn \CT@arc@ #1 #2
97   {
98     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
99     { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
100 }
```

Idem for `\CT@drs@`.

```
101   \cs_set_protected:Npn \CT@drsc@ { }
102   \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
103   \cs_set:Npn \CT@drs@ #1 #2
104   {
105     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
106     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
107   }
108   \cs_set:Npn \hline
109   {
110     \noalign { \ifnum 0 = ` } \fi
111     \cs_set_eq:NN \hskip \vskip
112     \cs_set_eq:NN \vrule \hrule
113     \cs_set_eq:NN \@width \@height
114     { \CT@arc@ \vline }
115     \futurelet \reserved@a
116     \@xhline
117   }
118 }
119 }
```

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.

```
120 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
121 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
122 {
123   \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
124   \int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
125   \multispan { \int_eval:n { #2 - #1 + 1 } }
126   {
127     \CT@arc@
128     \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`<sup>50</sup>

```
129   \skip_horizontal:N \c_zero_dim
130 }
```

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`.

```
131   \everycr { }
```

---

<sup>50</sup>See question 99041 on TeX StackExchange.

```

132 \cr
133 \noalign { \skip_vertical:N -\arrayrulewidth }
134 }

```

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.

```

135 \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`.

```

136 { \@@_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*.

```

137 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1-#2 \q_stop }
138 \cs_set:Npn \@@_cline_i:w #1-#2-#3 \q_stop
139 {

```

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).

```

140 \int_compare:nNnT { #1 } < { #2 }
141 { \multispan { \int_eval:n { #2 - #1 } } & }
142 \multispan { \int_eval:n { #3 - #2 + 1 } }
143 {
144 \CT@arc@
145 \leaders \hrule \@height \arrayrulewidth \hfill
146 \skip_horizontal:N \c_zero_dim
147 }

```

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

```

148 \peek_meaning_remove_ignore_spaces:NTF \cline
149 { & \@@_cline_i:en { \@@_succ:n { #3 } } }
150 { \everycr { } \cr }
151 }
152 \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.

```

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

```

The following command is a small shortcut.

```

155 \cs_new:Npn \@@_math_toggle_token:
156 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

```

```

157 \cs_new_protected:Npn \@@_set_CT@arc@:
158 { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
159 \cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
160 { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
161 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
162 { \cs_set:Npn \CT@arc@ { \color { #1 } } }

```

```

163 \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`.

```

164 \bool_new:N \c_@@_siunitx_loaded_bool
165 \AtBeginDocument
166 {
167 \ifpackageloaded { siunitx }
168 { \bool_set_true:N \c_@@_siunitx_loaded_bool }
169 { }

```

```
170 }
```

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

```
171 \AtBeginDocument
172 {
173   \bool_if:nTF { ! \c_@@_siunitx_loaded_bool }
174   { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
175 }
```

For version of `siunitx` at least equal to 3.0, the adaptation is different from previous ones. We test the version of `siunitx` by the existence of the control sequence `\siunitx_cell_begin:w`.

```
176   \cs_if_exist:nTF \siunitx_cell_begin:w
177   {
178     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
179     {
180       \renewcommand*{\NC@rewrite@S}[1] []
181       {
182         \@temptokena \exp_after:wN
183         {
184           \tex_the:D \@temptokena
185           > {
186             \@@_Cell:
187             \keys_set:nn { siunitx } { ##1 }
188             \siunitx_cell_begin:w
189           }
190         }
191       }
192     }
193   }
```

`\@@_true_c:` will be replaced statically by `c` at the end of the construction of the preamble.

```
190       \@@_true_c:
191       < { \siunitx_cell_end: \@@_end_Cell: }
192     }
193   \NC@find
194 }
195 }
196 {
197   \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
198   {
199     \renewcommand*{\NC@rewrite@S}[1] []
200     {
201       \@temptokena \exp_after:wN
202       {
203         \tex_the:D \@temptokena
204         > { \@@_Cell: \c_@@_table_collect_begin_tl S {##1} }
205         \@@_true_c:
206         < { \c_@@_table_print_tl \@@_end_Cell: }
207       }
208     }
209   \NC@find
210 }
211 }
212 }
213 }
214 }
```

The following code is used to define `\c_@@_table_collect_begin_tl` and `\c_@@_table_print_tl` when the version of `siunitx` is prior to v3.0. The command `\@@_adapt_S_column:` is used in the environment `{NiceArrayWithDelims}`.

```
215 \cs_set_protected:Npn \@@_adapt_S_column:
216 {
217   \bool_if:NT \c_@@_siunitx_loaded_bool
218   {
219     \group_begin:
220     \@temptokena = { }
221     \cs_set_eq:NN \NC@find \prg_do_nothing:
```

```

222 \NC@rewrite@S { }
223 \tl_gset:Nv \g_tmpa_tl \@temptokena
224 \group_end:
225 \tl_new:N \c_@@_table_collect_begin_tl
226 \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
227 \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
228 \tl_new:N \c_@@_table_print_tl
229 \tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl { -1 } }
230 \cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
231 }
232 }

```

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

```

233 \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.

```

234 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
235 {
236   \iow_now:Nn \@mainaux
237   {
238     \ExplSyntaxOn
239     \cs_if_free:NT \pgfsyspdfmark
240     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
241     \ExplSyntaxOff
242   }
243   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
244 }

```

## Parameters

For compatibility with versions prior to 5.0, we provide a load-time option `define_L_C_R`. With this option, it's possible to use the letters `L`, `C` and `R` instead of `l`, `c` and `r` in the preamble of the environments of `nicematrix` as it was mandatory before version 5.0.

```

245 \bool_new:N \c_@@_define_L_C_R_bool
246 \cs_new_protected:Npn \@@_define_L_C_R:
247 {
248   \newcolumntype L l
249   \newcolumntype C c
250   \newcolumntype R r
251 }

```

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.

```

252 \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).

```

253 \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.

```

254 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
255 { \int_use:N \g_@@_env_int }

```

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

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

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

```
258 \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).

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

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

```
260 \int_new:N \g_@@_row_total_int
261 \int_new:N \g_@@_col_total_int
```

The following token list will contain the type of the current cell (`l`, `c` or `r`). It will be used by the blocks.

```
262 \tl_new:N \l_@@_cell_type_tl
263 \tl_set:Nn \l_@@_cell_type_tl { 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`.

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

Idem pour the mono-row blocks.

```
265 \dim_new:N \g_@@_blocks_ht_dim
266 \dim_new:N \g_@@_blocks_dp_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`.

```
267 \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.

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

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

```
269 \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.

```
270 \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.

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

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

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

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

```

273 \bool_new:N \g_@@_rotate_bool

274 \cs_new_protected:Npn \@@_test_if_math_mode:
275 {
276   \if_mode_math: \else:
277     \@@_fatal:n { Outside-math-mode }
278   \fi:
279 }
```

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

```

280 \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.

```

281 \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”.

```

282 \colorlet { nicematrix-last-col } { . }
283 \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`).

```

284 \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*.

```

285 \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).

```

286 \cs_new:Npn \@@_full_name_env:
287 {
288   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
289     { command \space \c_backslash_str \g_@@_name_env_str }
290     { environment \space \{ \g_@@_name_env_str \} }
291 }
```

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`).

```

292 \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.

```

293 \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.

```

294 \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.

```
295 \int_new:N \l_@@_old_iRow_int
296 \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.

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

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.

```
298 \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).

```
299 \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).

```
300 \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`).

```
301 \tl_new:N \l_@@_code_before_tl
302 \bool_new:N \l_@@_code_before_bool
```

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

```
303 \dim_new:N \l_@@_x_initial_dim
304 \dim_new:N \l_@@_y_initial_dim
305 \dim_new:N \l_@@_x_final_dim
306 \dim_new:N \l_@@_y_final_dim
```

`expl3` 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`).

```
307 \dim_zero_new:N \l_tmpc_dim
308 \dim_zero_new:N \l_tmpd_dim
```

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

```
309 \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”.

```
310 \dim_new:N \g_@@_width_last_col_dim
311 \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.

```
312 \seq_new:N \g_@@_blocks_seq
```

We also manage a sequence of the *positions* of the blocks. Of course, it's redundant with the previous sequence, but it's for efficiency. In that sequence, each block is represented by only the four first components: `{imin}{jmin}{imax}{jmax}`.

```
313 \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 four components: `{imin}{jmin}{imax}{jmax}`.

```
314 \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.

```
315 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners` (or the key `hvlines-except-corners`), all the cells which are in an (empty) corner will be stored in the following sequence.

```
316 \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`).

```
317 \seq_new:N \g_@@_submatrix_names_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`).

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

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

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

```
321 \int_new:N \l_@@_col_max_int
```

The following sequence will be used when the command `\SubMatrix` is used in the `code-before` (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.

```
322 \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).

```
323 \int_new:N \g_@@_static_num_of_col_int
```

The following parameters correspond to the keys `fill`, `draw`, `borders` and `rounded-corners` of the command `\Block`.

```
324 \tl_new:N \l_@@_fill_tl
```

```
325 \tl_new:N \l_@@_draw_tl
```

```
326 \clist_new:N \l_@@_borders_clist
```

```
327 \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`.

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

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

```
329 \dim_new:N \l_@@_line_width_dim
```

The parameters of position of the label of a block. For the horizontal position, the possible values are `c`, `r` and `l`. 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`.

```
330 \tl_new:N \l_@@_hpos_of_block_tl
331 \tl_set:Nn \l_@@_hpos_of_block_tl { c }
332 \tl_new:N \l_@@_vpos_of_block_tl
333 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

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

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

```
335 \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.

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

337 \dim_new:N \l_@@_submatrix_extra_height_dim
338 \dim_new:N \l_@@_submatrix_left_xshift_dim
339 \dim_new:N \l_@@_submatrix_right_xshift_dim
340 \clist_new:N \l_@@_hlines_clist
341 \clist_new:N \l_@@_vlines_clist
342 \clist_new:N \l_@@_submatrix_hlines_clist
343 \clist_new:N \l_@@_submatrix_vlines_clist
```

## 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.

```
344 \int_new:N \l_@@_first_row_int
345 \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.

```
346 \int_new:N \l_@@_first_col_int
347 \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).

```
348 \int_new:N \l_@@_last_row_int
349 \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”.<sup>51</sup>

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

Idem for `\l_@@_last_col_without_value_bool`

```
351 \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.

```
352 \int_new:N \l_@@_last_col_int
353 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

```
\begin{pNiceArray}{cc}[last-col]
1 & 2 \\
3 & 4
\end{pNiceArray}
```

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

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

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

## 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.

```
355 \newcounter { tablarnote }
```

---

<sup>51</sup>We can’t use `\l_@@_last_row_int` for this usage because, if `nicematrix` has read its value from the `aux` file, the value of the counter won’t be `-1` any longer.

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

```
356 \seq_new:N \g_@@_tabularnotes_seq
```

However, before the actual tabular notes, it's possible to put a text specified by the key `tabularnote` of the environment. The token list `\l_@@_tabularnote_tl` corresponds to the value of that key.

```
357 \tl_new:N \l_@@_tabularnote_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.  $a,b,c$ ).

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

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

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

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

```
360 \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`.

```
361 \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`.

```
362 \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`).

```
363 \AtBeginDocument
364 {
365   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
366   {
367     \NewDocumentCommand \tabularnote { m }
368     { \@@_error:n { enumitem-not-loaded } }
369   }
370 }
```

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.

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

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.<sup>52</sup>

```

390     \NewDocumentCommand \tabularnote { m }
391     {
392         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
393         { \@@_error:n { tabularnote~forbidden } }
394         {

```

`\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.  $a,b,c$ ).

```

395         \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`.

```

396         \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
397         \peek_meaning:NF \tabularnote
398         {

```

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.

```

399         \hbox_set:Nn \l_tmpa_box
400         {

```

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

```

401         \@@_notes_label_in_tabular:n
402         {
403             \stepcounter { tabularnote }
404             \@@_notes_style:n { tabularnote }
405             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
406             {
407                 ,
408                 \stepcounter { tabularnote }
409                 \@@_notes_style:n { tabularnote }
410             }
411         }
412     }

```

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.

```

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

```

417         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
418         }
419     }
420 }

```

---

<sup>52</sup>We should try to find a solution to that problem.

```

421     }
422 }

```

## 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.

```

423 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
424 {
425   \begin { pgfscope }
426   \pgfset
427   {
428     outer~sep = \c_zero_dim ,
429     inner~sep = \c_zero_dim ,
430     minimum~size = \c_zero_dim
431   }
432   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
433   \pgfnode
434   { rectangle }
435   { center }
436   {
437     \vbox_to_ht:nn
438     { \dim_abs:n { #5 - #3 } }
439     {
440       \vfill
441       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
442     }
443   }
444   { #1 }
445   { }
446   \end { pgfscope }
447 }

```

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.

```

448 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
449 {
450   \begin { pgfscope }
451   \pgfset
452   {
453     outer~sep = \c_zero_dim ,
454     inner~sep = \c_zero_dim ,
455     minimum~size = \c_zero_dim
456   }
457   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
458   \pgfpointdiff { #3 } { #2 }
459   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
460   \pgfnode
461   { rectangle }
462   { center }
463   {
464     \vbox_to_ht:nn
465     { \dim_abs:n \l_tmpb_dim }
466     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
467   }
468   { #1 }
469   { }
470   \end { pgfscope }
471 }

```

## 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.

```
472 \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`.

```
473 \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`).

```
474 \dim_new:N \l_@@_cell_space_top_limit_dim
475 \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.

```
476 \dim_new:N \l_@@_inter_dots_dim
477 \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).

```
478 \dim_new:N \l_@@_xdots_shorten_dim
479 \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.

```
480 \dim_new:N \l_@@_radius_dim
481 \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.

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

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

```
485 \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).

```
486 \tl_new:N \l_@@_baseline_tl
487 \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`).

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

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

```
489 \bool_new:N \l_@@_parallelize_diags_bool
490 \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.

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

```
492 \dim_new:N \l_@@_notes_above_space_dim
493 \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.).

```
494 \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).

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

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

```
496 \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.

```
497 \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”.

```
498 \bool_new:N \l_@@_medium_nodes_bool
499 \bool_new:N \l_@@_large_nodes_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`).

```
500 \dim_new:N \l_@@_left_margin_dim
501 \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`.

```
502 \dim_new:N \l_@@_extra_left_margin_dim
503 \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.

```
504 \tl_new:N \l_@@_end_of_row_tl
505 \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 “:”.

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

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

```
507 \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.

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

```
509 \keys_define:nn { NiceMatrix / xdots }
510 {
511   line-style .code:n =
512   {
513     \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.

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

```
524   down .tl_set:N = \l_@@_xdots_down_tl ,
525   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).

```
526   draw-first .code:n = \prg_do_nothing: ,
527   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
528 }
```

```
529 \keys_define:nn { NiceMatrix / rules }
530 {
531   color .tl_set:N = \l_@@_rules_color_tl ,
532   color .value_required:n = true ,
533   width .dim_set:N = \arrayrulewidth ,
534   width .value_required:n = true
535 }
```

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

```
536 \keys_define:nn { NiceMatrix / Global }
537 {
```

```

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

```

589 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
590 renew-dots .value_forbidden:n = true ,
591 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
592 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
593 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
594 create-extra-nodes .meta:n =
595   { create-medium-nodes , create-large-nodes } ,
596 left-margin .dim_set:N = \l_@@_left_margin_dim ,
597 left-margin .default:n = \arraycolsep ,

```

```

598 right-margin .dim_set:N = \l_@@_right_margin_dim ,
599 right-margin .default:n = \arraycolsep ,
600 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
601 margin .default:n = \arraycolsep ,
602 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
603 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
604 extra-margin .meta:n =
605   { extra-left-margin = #1 , extra-right-margin = #1 } ,
606 extra-margin .value_required:n = true ,
607 }

```

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

```

608 \keys_define:nn { NiceMatrix / Env }
609 {
610

```

The key `hvlines-except-corners` is now deprecated.

```

611 hvlines-except-corners .code:n =
612   {
613     \clist_set:Nn \l_@@_corners_clist { #1 }
614     \clist_set:Nn \l_@@_vlines_clist { all }
615     \clist_set:Nn \l_@@_hlines_clist { all }
616   } ,
617 hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
618 corners .clist_set:N = \l_@@_corners_clist ,
619 corners .default:n = { NW , SW , NE , SE } ,
620 code-before .code:n =
621   {
622     \tl_if_empty:nF { #1 }
623     {
624       \tl_put_right:Nn \l_@@_code_before_tl { #1 }
625       \bool_set_true:N \l_@@_code_before_bool
626     }
627   } ,

```

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

```

628 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
629 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
630 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
631 baseline .tl_set:N = \l_@@_baseline_tl ,
632 baseline .value_required:n = true ,
633 columns-width .code:n =
634   \tl_if_eq:nnTF { #1 } { auto }
635     { \bool_set_true:N \l_@@_auto_columns_width_bool }
636     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
637 columns-width .value_required:n = true ,
638 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.

```

639 \legacy_if:nF { measuring@ }
640 {
641   \str_set:Nn \l_tmpa_str { #1 }
642   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
643     { @@_error:nn { Duplicate-name } { #1 } }
644     { \seq_gput_left:N \g_@@_names_seq \l_tmpa_str }
645   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
646 } ,
647 name .value_required:n = true ,
648 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
649 code-after .value_required:n = true ,

```

```

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

```

713 }

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

```

714 \keys_define:nn { NiceMatrix }
715 {
716   NiceMatrixOptions .inherit:n =
717     { NiceMatrix / Global } ,
718   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
719   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
720   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
721   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
722   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
723   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
724   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
725   NiceMatrix .inherit:n =
726     {
727       NiceMatrix / Global ,
728       NiceMatrix / Env ,
729     } ,
730   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
731   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
732   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
733   NiceTabular .inherit:n =
734     {
735       NiceMatrix / Global ,
736       NiceMatrix / Env
737     } ,
738   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
739   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
740   NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
741   NiceArray .inherit:n =
742     {
743       NiceMatrix / Global ,
744       NiceMatrix / Env ,
745     } ,
746   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
747   NiceArray / rules .inherit:n = NiceMatrix / rules ,
748   NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
749   pNiceArray .inherit:n =
750     {
751       NiceMatrix / Global ,
752       NiceMatrix / Env ,
753     } ,
754   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
755   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
756   pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
757 }
```

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

```

758 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
759 {
760   last-col .code:n = \tl_if_empty:nF { #1 }
761     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
762     \int_zero:N \l_@@_last_col_int ,
763   small .bool_set:N = \l_@@_small_bool ,
764   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.

```

765   renew-matrix .code:n = \@@_renew_matrix: ,
766   renew-matrix .value_forbidden:n = true ,
```

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

```

767 transparent .code:n =
768 {
769   \@@_renew_matrix:
770   \bool_set_true:N \l_@@_renew_dots_bool
771   \@@_error:n { Key~transparent }
772 } ,
773 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}`.

```

774 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.

```

775 columns-width .code:n =
776   \tl_if_eq:nnTF { #1 } { auto }
777   { \@@_error:n { Option~auto~for~columns~width } }
778   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

779 allow-duplicate-names .code:n =
780   \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
781 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`).

```

782 letter-for-dotted-lines .code:n =
783 {
784   \tl_if_single_token:nTF { #1 }
785   { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
786   { \@@_error:n { One~letter~allowed } }
787 } ,
788 letter-for-dotted-lines .value_required:n = true ,
789 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
790 notes .value_required:n = true ,
791 sub-matrix .code:n =
792   \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
793 sub-matrix .value_required:n = true ,
794 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
795 }
796 \str_new:N \l_@@_letter_for_dotted_lines_str
797 \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.

```

798 \NewDocumentCommand \NiceMatrixOptions { m }
799 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

800 \keys_define:nn { NiceMatrix / NiceMatrix }
801 {
802   last-col .code:n = \tl_if_empty:nTF {#1}

```

```

803         {
804             \bool_set_true:N \l_@@_last_col_without_value_bool
805             \int_set:Nn \l_@@_last_col_int { -1 }
806         }
807         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
808     l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
809     r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
810     small .bool_set:N = \l_@@_small_bool ,
811     small .value_forbidden:n = true ,
812     unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
813 }

```

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

```

814 \keys_define:nn { NiceMatrix / NiceArray }
815 {

```

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.

```

816     small .bool_set:N = \l_@@_small_bool ,
817     small .value_forbidden:n = true ,
818     last-col .code:n = \tl_if_empty:nF { #1 }
819         { \@@_error:n { last-col~non-empty~for~NiceArray } }
820         \int_zero:N \l_@@_last_col_int ,
821     notes / para .bool_set:N = \l_@@_notes_para_bool ,
822     notes / para .default:n = true ,
823     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
824     notes / bottomrule .default:n = true ,
825     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
826     tabularnote .value_required:n = true ,
827     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
828     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
829     unknown .code:n = \@@_error:n { Unknown~option~for~NiceArray }
830 }
831 \keys_define:nn { NiceMatrix / pNiceArray }
832 {
833     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
834     last-col .code:n = \tl_if_empty:nF { #1 }
835         { \@@_error:n { last-col~non-empty~for~NiceArray } }
836         \int_zero:N \l_@@_last_col_int ,
837     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
838     small .bool_set:N = \l_@@_small_bool ,
839     small .value_forbidden:n = true ,
840     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
841     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
842     unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
843 }

```

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

```

844 \keys_define:nn { NiceMatrix / NiceTabular }
845 {
846     notes / para .bool_set:N = \l_@@_notes_para_bool ,
847     notes / para .default:n = true ,
848     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
849     notes / bottomrule .default:n = true ,
850     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
851     tabularnote .value_required:n = true ,
852     last-col .code:n = \tl_if_empty:nF { #1 }
853         { \@@_error:n { last-col~non-empty~for~NiceArray } }
854         \int_zero:N \l_@@_last_col_int ,

```



```

855   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
856   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
857   unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular }
858 }

```

## Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_Cell:-\@@_end_Cell:` 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}`).

```

859 \cs_new_protected:Npn \@@_Cell:
860 {

```

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

```

861   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n

```

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

```

862   \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.

```

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

```

The content of the cell is composed in the box `\l_@@_cell_box` because we want to compute some dimensions of the box. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the `\@@_end_Cell:` (and the potential `\c_math_toggle_token` also).

```

865   \hbox_set:Nw \l_@@_cell_box
866   \bool_if:NF \l_@@_NiceTabular_bool
867   {
868     \c_math_toggle_token
869     \bool_if:NT \l_@@_small_bool \scriptstyle
870   }

```

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.

```

871   \int_compare:nNnTF \c@iRow = 0
872   {
873     \int_compare:nNnT \c@jCol > 0
874     {
875       \l_@@_code_for_first_row_tl
876       \xglobal \colorlet { nicematrix-first-row } { . }
877     }
878   }
879   {
880     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
881     {
882       \l_@@_code_for_last_row_tl
883       \xglobal \colorlet { nicematrix-last-row } { . }
884     }
885   }
886 }

```

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.

```

887 \cs_new_protected:Npn \@@_begin_of_row:
888 {

```

```

889 \int_gincr:N \c@iRow
890 \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
891 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
892 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
893 \pgfpicture
894 \pgfrememberpicturepositiononpagetrue
895 \pgfcoordinate
896 { \@@_env: - row - \int_use:N \c@iRow - base }
897 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
898 \str_if_empty:NF \l_@@_name_str
899 {
900   \pgfnodealias
901   { \l_@@_name_str - row - \int_use:N \c@iRow - base }
902   { \@@_env: - row - \int_use:N \c@iRow - base }
903 }
904 \endpgfpicture
905 }

```

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.

```

906 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
907 {
908   \int_compare:nNnTF \c@iRow = 0
909   {
910     \dim_gset:Nn \g_@@_dp_row_zero_dim
911     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
912     \dim_gset:Nn \g_@@_ht_row_zero_dim
913     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
914   }
915   {
916     \int_compare:nNnT \c@iRow = 1
917     {
918       \dim_gset:Nn \g_@@_ht_row_one_dim
919       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
920     }
921   }
922 }
923 \cs_new_protected:Npn \@@_rotate_cell_box:
924 {
925   \box_rotate:Nn \l_@@_cell_box { 90 }
926   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
927   {
928     \vbox_set_top:Nn \l_@@_cell_box
929     {
930       \vbox_to_zero:n { }
931       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
932       \box_use:N \l_@@_cell_box
933     }
934   }
935   \bool_gset_false:N \g_@@_rotate_bool
936 }
937 \cs_new_protected:Npn \@@_adjust_size_box:
938 {
939   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
940   {
941     \box_set_wd:Nn \l_@@_cell_box
942     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
943     \dim_gzero:N \g_@@_blocks_wd_dim
944   }
945   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim

```

```

946 {
947   \box_set_dp:Nn \l_@@_cell_box
948   { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
949   \dim_gzero:N \g_@@_blocks_dp_dim
950 }
951 \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
952 {
953   \box_set_ht:Nn \l_@@_cell_box
954   { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
955   \dim_gzero:N \g_@@_blocks_ht_dim
956 }
957 }
958 \cs_new_protected:Npn \@@_end_Cell:
959 {
960   \@@_math_toggle_token:
961   \hbox_set_end:
962   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
963   \@@_adjust_size_box:
964   \box_set_ht:Nn \l_@@_cell_box
965   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
966   \box_set_dp:Nn \l_@@_cell_box
967   { \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”).

```

968   \dim_gset:Nn \g_@@_max_cell_width_dim
969   { \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”.

```

970   \@@_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.

```

971   \bool_if:NTF \g_@@_empty_cell_bool
972   { \box_use_drop:N \l_@@_cell_box }
973   {
974     \bool_lazy_or:nnTF
975     \g_@@_not_empty_cell_bool
976     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
977     \@@_node_for_cell:
978     { \box_use_drop:N \l_@@_cell_box }
979   }
980   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
981   \bool_gset_false:N \g_@@_empty_cell_bool
982   \bool_gset_false:N \g_@@_not_empty_cell_bool
983 }

```

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

```

984 \cs_new_protected:Npn \@@_node_for_cell:
985 {
986   \pgfpicture
987   \pgfsetbaseline \c_zero_dim
988   \pgfrememberpicturepositiononpagetrue
989   \pgfset
990   {
991     inner~sep = \c_zero_dim ,
992     minimum~width = \c_zero_dim
993   }
994   \pgfnode
995   { rectangle }
996   { base }
997   { \box_use_drop:N \l_@@_cell_box }
998   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
999   { }
1000   \str_if_empty:NF \l_@@_name_str
1001   {
1002     \pgfnodealias
1003     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1004     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1005   }
1006   \endpgfpicture
1007 }

```

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.

```

1008 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1009 {
1010   \cs_new_protected:Npn \@@_patch_node_for_cell:
1011   {
1012     \hbox_set:Nn \l_@@_cell_box
1013     {
1014       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1015       \hbox_overlap_left:n
1016       {
1017         \pgfsys@markposition
1018         { \@@_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.

```

1019       #1
1020     }
1021     \box_use:N \l_@@_cell_box
1022     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1023     \hbox_overlap_left:n
1024     {
1025       \pgfsys@markposition
1026       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1027       #1
1028     }
1029   }
1030 }
1031 }
1032 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1033 {
1034   \@@_patch_node_for_cell:n
1035   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1036 }
1037 { \@@_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.

```
1038 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1039 {
1040   \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1041   { \g_@@_#2 _ lines _ tl }
1042   {
1043     \use:c { @@ _ draw _ #2 : nnn }
1044     { \int_use:N \c@iRow }
1045     { \int_use:N \c@jCol }
1046     { \exp_not:n { #3 } }
1047   }
1048 }
```

We want to use `\array` of `array`. However, if the class used is `revtex4-1` or `revtex4-2`, we have to do some tuning and use the command `\@array@array` instead of `\array` because these classes do a redefinition of `\array` incompatible with our use of `\array`.

```
1049 \cs_new_protected:Npn \@@_revtex_array:
1050 {
1051   \cs_set_eq:NN \@acol1 \@arrayacol
1052   \cs_set_eq:NN \@acolr \@arrayacol
1053   \cs_set_eq:NN \@acol \@arrayacol
1054   \cs_set_nopar:Npn \@halignto { }
1055   \@array@array
1056 }
1057 \cs_new_protected:Npn \@@_array:
1058 {
1059   \bool_if:NTF \c_@@_revtex_bool
1060   \@@_revtex_array:
1061   {
1062     \bool_if:NTF \l_@@_NiceTabular_bool
1063     { \dim_set_eq:NN \col@sep \tabcolsep }
1064     { \dim_set_eq:NN \col@sep \arraycolsep }
1065     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1066     { \cs_set_nopar:Npn \@halignto { } }
1067     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
1068 }
```

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

```
1068   \@tabarray
1069 }
```

`\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.

```
1070 [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1071 }
```

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.

```
1072 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1073 \cs_new_protected:Npn \@@_create_row_node:
1074 {
```

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

```
1075   \hbox
1076   {
1077     \bool_if:NT \l_@@_code_before_bool
1078     {
1079       \vtop
1080       {
1081         \skip_vertical:N 0.5\arrayrulewidth
1082         \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
1083         \skip_vertical:N -0.5\arrayrulewidth
1084       }
1085     }
1086     \pgfpicture
1087     \pgfrememberpicturepositiononpagetrue
1088     \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
1089     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1090     \str_if_empty:NF \l_@@_name_str
1091     {
1092       \pgfnodealias
1093       { \l_@@_name_str - row - \int_use:N \c@iRow }
1094       { \@@_env: - row - \int_use:N \c@iRow }
1095     }
1096     \endpgfpicture
1097   }
1098 }
```

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

```
1099 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1100 \cs_new_protected:Npn \@@_everycr_i:
1101 {
1102   \int_gzero:N \c@jCol
1103   \bool_gset_false:N \g_@@_after_col_zero_bool
1104   \bool_if:NF \g_@@_row_of_col_done_bool
1105   {
1106     \@@_create_row_node:
```

We don't draw the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for theses rules.

```
1107   \tl_if_empty:NF \l_@@_hlines_clist
1108   {
1109     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1110     {
1111       \exp_args:NNx
1112       \clist_if_in:NnT
1113       \l_@@_hlines_clist
1114       { \@@_succ:n \c@iRow }
1115     }
1116   }
```

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.

```
1117   \int_compare:nNnT \c@iRow > { -1 }
1118   {
1119     \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@`.

```

1120             { \hrule height \arrayrulewidth width \c_zero_dim }
1121         }
1122     }
1123 }
1124 }
1125 }

```

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`).

```

1126 \cs_set_protected:Npn \@@_newcolumntype #1
1127 {
1128     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1129     \peek_meaning:NTF [
1130         { \newcol@ #1 }
1131         { \newcol@ #1 [ 0 ] }
1132     }

```

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

```

1133 \cs_set_protected:Npn \@@_renew_dots:
1134 {
1135     \cs_set_eq:NN \ldots \@@_Ldots
1136     \cs_set_eq:NN \cdots \@@_Cdots
1137     \cs_set_eq:NN \vdots \@@_Vdots
1138     \cs_set_eq:NN \ddots \@@_Ddots
1139     \cs_set_eq:NN \iddots \@@_Iddots
1140     \cs_set_eq:NN \dots \@@_Ldots
1141     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1142 }

```

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

```

1143 \cs_new_protected:Npn \@@_colortbl_like:
1144 {
1145     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1146     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1147     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1148 }

```

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

```

1149 \cs_new_protected:Npn \@@_pre_array_ii:
1150 {

```

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 <sup>53</sup>.

```

1151     \bool_if:NT \c_@@_booktabs_loaded_bool
1152     { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1153     \box_clear_new:N \l_@@_cell_box
1154     \cs_if_exist:NT \theiRow
1155     { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }

```

---

<sup>53</sup>cf. `\nicematrix@redefine@check@rerun`

```

1156 \int_gzero_new:N \c@iRow
1157 \cs_if_exist:NT \thejCol
1158 { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1159 \int_gzero_new:N \c@jCol
1160 \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}`).

```

1161 \bool_if:NT \l_@@_small_bool
1162 {
1163     \cs_set_nopar:Npn \arraystretch { 0.47 }
1164     \dim_set:Nn \arraycolsep { 1.45 pt }
1165 }

```

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`.

```

1166 \cs_set_nopar:Npn \ialign
1167 {
1168     \bool_if:NTF \c_@@_colortbl_loaded_bool
1169     {
1170         \CT@everycr
1171         {
1172             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1173             \@@_everycr:
1174         }
1175     }
1176     { \everycr { \@@_everycr: } }
1177     \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`<sup>54</sup> 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`.

```

1178 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1179 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1180 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1181 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1182 \dim_gzero_new:N \g_@@_ht_row_one_dim
1183 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1184 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1185 \dim_gzero_new:N \g_@@_ht_last_row_dim
1186 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1187 \dim_gzero_new:N \g_@@_dp_last_row_dim
1188 \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`.

```

1189 \cs_set_eq:NN \ialign \@@_old_ialign:
1190 \halign
1191 }

```

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).

```

1192 \cs_set_eq:NN \@@_old_ldots \ldots

```

---

<sup>54</sup>The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.



```

1193 \cs_set_eq:NN \@_old_cdots \cdots
1194 \cs_set_eq:NN \@_old_vdots \vdots
1195 \cs_set_eq:NN \@_old_ddots \ddots
1196 \cs_set_eq:NN \@_old_iddots \iddots
1197 \bool_if:NTF \l_@_standard_cline_bool
1198 { \cs_set_eq:NN \cline \@_standard_cline }
1199 { \cs_set_eq:NN \cline \@_cline }
1200 \cs_set_eq:NN \Ldots \@_Ldots
1201 \cs_set_eq:NN \Cdots \@_Cdots
1202 \cs_set_eq:NN \Vdots \@_Vdots
1203 \cs_set_eq:NN \Ddots \@_Ddots
1204 \cs_set_eq:NN \Iddots \@_Iddots
1205 \cs_set_eq:NN \hdottedline \@_hdottedline:
1206 \cs_set_eq:NN \Hline \@_Hline:
1207 \cs_set_eq:NN \Hspace \@_Hspace:
1208 \cs_set_eq:NN \Hdotsfor \@_Hdotsfor:
1209 \cs_set_eq:NN \Vdotsfor \@_Vdotsfor:
1210 \cs_set_eq:NN \multicolumn \@_multicolumn:nnn
1211 \cs_set_eq:NN \Block \@_Block:
1212 \cs_set_eq:NN \rotate \@_rotate:
1213 \cs_set_eq:NN \OnlyMainNiceMatrix \@_OnlyMainNiceMatrix:n
1214 \cs_set_eq:NN \dotfill \@_old_dotfill:
1215 \cs_set_eq:NN \CodeAfter \@_CodeAfter:
1216 \cs_set_eq:NN \diagbox \@_diagbox:nn
1217 \cs_set_eq:NN \NotEmpty \@_NotEmpty:
1218 \bool_if:NT \l_@_colortbl_like_bool \@_colortbl_like:
1219 \bool_if:NT \l_@_renew_dots_bool \@_renew_dots:

```

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

```

1220 \seq_gclear_new:N \g_@_multicolumn_cells_seq
1221 \seq_gclear_new: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).

```

1222 \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`).

```

1223 \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:` executed at the beginning of each cell.

```

1224 \int_gzero_new:N \g_@_col_total_int
1225 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1226 \@_renew_NC@rewrite@S:
1227 \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.

```

1228 \tl_gclear_new:N \g_@_Cdots_lines_tl
1229 \tl_gclear_new:N \g_@_Ldots_lines_tl
1230 \tl_gclear_new:N \g_@_Vdots_lines_tl
1231 \tl_gclear_new:N \g_@_Ddots_lines_tl
1232 \tl_gclear_new:N \g_@_Iddots_lines_tl
1233 \tl_gclear_new:N \g_@_HVdotsfor_lines_tl

1234 \tl_gclear_new:N \g_nicematrix_code_before_tl
1235 }

```

This is the end of \l\_@@\_pre\_array\_ii:.

```

1236 \cs_new_protected:Npn \l_@@_pre_array:
1237 {
1238   \seq_gclear:N \g_@@_submatrix_seq
1239   \bool_if:NT \l_@@_code_before_bool \l_@@_exec_code_before:

```

A value of  $-1$  for the counter  $\l_@@\_last\_row\_int$  means that the user has used the option `last-row` without value, that is to say without specifying the number of that last row. In this case, we try to read that value from the aux file (if it has been written on a previous run).

```

1240   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1241   {
1242     \tl_put_right:Nn \l_@@_update_for_first_and_last_row:
1243     {
1244       \dim_gset:Nn \g_@@_ht_last_row_dim
1245       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1246       \dim_gset:Nn \g_@@_dp_last_row_dim
1247       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1248     }
1249   }
1250   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1251   {
1252     \bool_set_true:N \l_@@_last_row_without_value_bool

```

A value based on the name is more reliable than a value based on the number of the environment.

```

1253   \str_if_empty:NTF \l_@@_name_str
1254   {
1255     \cs_if_exist:cT { @@_last_row_ \int_use:N \g_@@_env_int }
1256     {
1257       \int_set:Nn \l_@@_last_row_int
1258       { \use:c { @@_last_row_ \int_use:N \g_@@_env_int } }
1259     }
1260   }
1261   {
1262     \cs_if_exist:cT { @@_last_row_ \l_@@_name_str }
1263     {
1264       \int_set:Nn \l_@@_last_row_int
1265       { \use:c { @@_last_row_ \l_@@_name_str } }
1266     }
1267   }
1268 }

```

A value of  $-1$  for the counter  $\l_@@\_last\_col\_int$  means that the user has used the option `last-col` without value, that is to say without specifying the number of that last column. In this case, we try to read that value from the aux file (if it has been written on a previous run).

```

1269   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1270   {
1271     \str_if_empty:NTF \l_@@_name_str
1272     {
1273       \cs_if_exist:cT { @@_last_col_ \int_use:N \g_@@_env_int }
1274       {
1275         \int_set:Nn \l_@@_last_col_int
1276         { \use:c { @@_last_col_ \int_use:N \g_@@_env_int } }
1277       }
1278     }
1279     {
1280       \cs_if_exist:cT { @@_last_col_ \l_@@_name_str }
1281       {
1282         \int_set:Nn \l_@@_last_col_int
1283         { \use:c { @@_last_col_ \l_@@_name_str } }
1284       }
1285     }
1286   }

```

The code in `\@@_pre_array_ii:` is used only by `{NiceArrayWithDelims}`.

```
1287 \@@_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.

```
1288 \box_clear_new:N \l_@@_the_array_box
```

If the user has loaded `nicematrix` with the option `define-L-C-R`, he will be able to use `L`, `C` and `R` instead of `l`, `c` and `r` in the preambles of the environments of `nicematrix` (it's a compatibility mode since `L`, `C` and `R` were mandatory before version 5.0).

```
1289 \bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:
```

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

```
1290 \@@_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]`).

```
1291 \dim_zero_new:N \l_@@_left_delim_dim
1292 \dim_zero_new:N \l_@@_right_delim_dim
1293 \bool_if:NTF \l_@@_NiceArray_bool
1294 {
1295   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1296   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1297 }
1298 {
```

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

```
1299 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1300 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1301 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1302 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1303 }
```

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).

```
1304 \hbox_set:Nw \l_@@_the_array_box
1305 \skip_horizontal:N \l_@@_left_margin_dim
1306 \skip_horizontal:N \l_@@_extra_left_margin_dim
1307 \c_math_toggle_token
1308 \bool_if:NTF \l_@@_light_syntax_bool
1309 { \use:c { @@-light-syntax } }
1310 { \use:c { @@-normal-syntax } }
1311 }
```

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

```
1312 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1313 {
1314   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1315   \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...

```
1316 \@@_pre_array:
1317 }
```

## The \CodeBefore

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

```
1318 \cs_new_protected:Npn \@@_pre_code_before:
1319 {
```

First, we give values to the LaTeX counters iRow and jCol. We remind that, in the code-before (and in the \CodeAfter) they represent the numbers of rows and columns of the array (without the potential last row and last column).

```
1320 \int_zero_new:N \c@iRow
1321 \int_set:Nn \c@iRow
1322 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
1323 \int_zero_new:N \c@jCol
1324 \int_set:Nn \c@jCol
1325 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 }
```

We have to adjust the values of \c@iRow and \c@jCol to take into account the potential last row and last column. A value of -2 for \l\_@@\_last\_row\_int means that there is no last row. Idem for the columns.

```
1326 \int_compare:nNf \l_@@_last_row_int = { -2 }
1327 { \int_decr:N \c@iRow }
1328 \int_compare:nNf \l_@@_last_col_int = { -2 }
1329 { \int_decr:N \c@jCol }
```

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.

```
1330 \pgfsys@markposition { \@@_env: - position }
1331 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1332 \pgfpicture
1333 \pgf@relevantforpicturesizefalse
```

First, the recreation of the row nodes.

```
1334 \int_step_inline:nnn
1335 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 1 }
1336 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 + 1 }
1337 {
1338 \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1339 \pgfcoordinate { \@@_env: - row - ##1 }
1340 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1341 }
```

Now, the recreation of the col nodes.

```
1342 \int_step_inline:nnn
1343 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
1344 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 + 1 }
1345 {
1346 \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1347 \pgfcoordinate { \@@_env: - col - ##1 }
1348 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1349 }
```

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

```
1350 \@@_create_diag_nodes:
```

Now, the creation of the cell nodes (i-j).

```
1351 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1352 \endpgfpicture

1353 \bool_if:NT \c_@@_tikz_loaded_bool
1354 {
1355 \tikzset
1356 {
1357 every-picture / .style =
1358 { overlay , name-prefix = \@@_env: - }
1359 }
1360 }
```

```

1361 \cs_set_eq:NN \cellcolor \@@_cellcolor
1362 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1363 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1364 \cs_set_eq:NN \rowcolor \@@_rowcolor
1365 \cs_set_eq:NN \rowcolors \@@_rowcolors
1366 \cs_set_eq:NN \arraycolor \@@_arraycolor
1367 \cs_set_eq:NN \columncolor \@@_columncolor
1368 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1369 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before

```

The list of the cells which are in the (empty) corners is stored in the `aux` file because we have to know it before the execution of the `\CodeBefore` (the commands which color the cells, rows and columns won't color the cells which are in the corners).

```

1370 \seq_if_exist:cT
1371 { c_@@_corners_cells_ \int_use:N \g_@@_env_int _ seq }
1372 {
1373   \seq_set_eq:Nc \l_@@_corners_cells_seq
1374   { c_@@_corners_cells_ \int_use:N \g_@@_env_int _ seq }
1375 }
1376 }

```

```

1377 \cs_new_protected:Npn \@@_exec_code_before:
1378 {
1379   \seq_gclear_new:N \g_@@_colors_seq
1380   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1381   \group_begin:

```

We compose the `code-before` in math mode in order to nullify the spaces put by the user between instructions in the `code-before`.

```

1382 \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`.

```

1383 \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.

```

1384 \@@_actually_color:
1385 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1386 \group_end:
1387 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1388 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1389 }

```

```

1390 \keys_define:nn { NiceMatrix / CodeBefore }
1391 {
1392   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1393   create-cell-nodes .default:n = true ,
1394   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1395   sub-matrix .value_required:n = true ,
1396   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1397   delimiters / color .value_required:n = true ,
1398   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1399 }
1400 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1401 {
1402   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1403   \@@_CodeBefore:w
1404 }

```

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.

```

1405 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1406 {
1407   \seq_if_exist:cT { @@_size_ \int_use:N \g_@@_env_int _ seq }
1408   {
1409     \@@_pre_code_before:
1410     #1
1411   }
1412 }

```

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.

```

1413 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1414 {
1415   \int_step_inline:nnn
1416   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 1 }
1417   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
1418   {
1419     \int_step_inline:nnn
1420     { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
1421     { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 }
1422     {
1423       \cs_if_exist:cT
1424       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1425       {
1426         \pgfsys@getposition
1427         { \@@_env: - ##1 - #####1 - NW }
1428         \@@_node_position:
1429         \pgfsys@getposition
1430         { \@@_env: - ##1 - #####1 - SE }
1431         \@@_node_position_i:
1432         \@@_pgf_rect_node:nnn
1433         { \@@_env: - ##1 - #####1 }
1434         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1435         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1436       }
1437     }
1438   }
1439 }

```

## The environment `{NiceArrayWithDelims}`

```

1440 \NewDocumentEnvironment { NiceArrayWithDelims }
1441 { m m O { } m ! O { } t \CodeBefore }
1442 {
1443   \@@_provide_pgfsyspdfmark:
1444   \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 exposant to a matrix in a mathematical formula.

```

1445   \bgroup

1446   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1447   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1448   \tl_gset:Nn \g_@@_preamble_tl { #4 }

```

```

1449 \int_gzero:N \g_@@_block_box_int
1450 \dim_zero:N \g_@@_width_last_col_dim
1451 \dim_zero:N \g_@@_width_first_col_dim
1452 \bool_gset_false:N \g_@@_row_of_col_done_bool
1453 \str_if_empty:NT \g_@@_name_env_str
1454 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }

```

The following line will be deleted when we will consider that only versions of siunitx after v3.0 are compatible with nicematrix.

```

1455 \@@_adapt_S_column:
1456 \bool_if:NTF \l_@@_NiceTabular_bool
1457 { \mode_leave_vertical:
1458   \@@_test_if_math_mode:
1459   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1460   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>55</sup>. 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.

```

1461 \cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@

```

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

```

1462 \cs_if_exist:NT \tikz@library@external@loaded
1463 {
1464   \tikzexternaldisable
1465   \cs_if_exist:NT \ifstandalone
1466   { \tikzset { external / optimize = false } }
1467 }

```

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

```

1468 \int_gincr:N \g_@@_env_int
1469 \bool_if:NF \l_@@_block_auto_columns_width_bool
1470 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

The sequence `\g_@@_blocks_seq` will contain the carateristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks. Of course, this is redundant but it's for efficiency.

```

1471 \seq_gclear:N \g_@@_blocks_seq
1472 \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`.

```

1473 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1474 \seq_gclear:N \g_@@_pos_of_xdots_seq

1475 \tl_if_exist:cT { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
1476 {
1477   \bool_set_true:N \l_@@_code_before_bool
1478   \exp_args:NNv \tl_put_right:Nn \l_@@_code_before_tl
1479   { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
1480 }

```

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`.

```

1481 \bool_if:NTF \l_@@_NiceArray_bool
1482 { \keys_set:nn { NiceMatrix / NiceArray } }
1483 { \keys_set:nn { NiceMatrix / pNiceArray } }
1484 { #3 , #5 }

```

---

<sup>55</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```

1485 \tl_if_empty:NF \l_@@_rules_color_tl
1486 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
1487 % \bigskip

```

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 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:.

```

1488 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
1489 }
1490 {
1491   \bool_if:NTF \l_@@_light_syntax_bool
1492     { \use:c { end @@-light-syntax } }
1493     { \use:c { end @@-normal-syntax } }
1494   \c_math_toggle_token
1495   \skip_horizontal:N \l_@@_right_margin_dim
1496   \skip_horizontal:N \l_@@_extra_right_margin_dim
1497   \hbox_set_end:

```

End of the construction of the array (in the box \l\_@@\_the\_array\_box).

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).

```

1498 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1499 {
1500   \bool_if:NF \l_@@_last_row_without_value_bool
1501   {
1502     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1503     {
1504       \@@_error:n { Wrong~last~row }
1505       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1506     }
1507   }
1508 }

```

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”.<sup>56</sup>

```

1509 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1510 \bool_if:nTF \g_@@_last_col_found_bool
1511 { \int_gdecr:N \c@jCol }
1512 {
1513   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1514   { \@@_error:n { last~col~not~used } }
1515 }

```

We fix also the value of \c@iRow and \g\_@@\_row\_total\_int with the same principle.

```

1516 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1517 \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. 106).

```

1518 \int_compare:nNnT \l_@@_first_col_int = 0
1519 {
1520   \skip_horizontal:N \col@sep
1521   \skip_horizontal:N \g_@@_width_first_col_dim
1522 }

```

---

<sup>56</sup>We remind that the potential “first column” (exterior) has the number 0.



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.

Remark that, in all cases, `@@_use_arraybox_with_notes_c:` is used.

```

1523   \bool_if:NTF \l_@@_NiceArray_bool
1524   {
1525     \str_case:VnF \l_@@_baseline_tl
1526     {
1527       b \@@_use_arraybox_with_notes_b:
1528       c \@@_use_arraybox_with_notes_c:
1529     }
1530     \@@_use_arraybox_with_notes:
1531   }

```

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).

```

1532   {
1533     \int_compare:nNnTF \l_@@_first_row_int = 0
1534     {
1535       \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1536       \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1537     }
1538     { \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”.<sup>57</sup>

```

1539     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1540     {
1541       \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1542       \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1543     }
1544     { \dim_zero:N \l_tmpb_dim }
1545     \hbox_set:Nn \l_tmpa_box
1546     {
1547       \c_math_toggle_token
1548       \tl_if_empty:NF \l_@@_delimiters_color_tl
1549       { \color { \l_@@_delimiters_color_tl } }
1550       \exp_after:wN \left \g_@@_left_delim_tl
1551       \vcenter
1552       {

```

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.

```

1553         \skip_vertical:N -\l_tmpa_dim
1554         \hbox
1555         {
1556           \bool_if:NTF \l_@@_NiceTabular_bool
1557           { \skip_horizontal:N -\tabcolsep }
1558           { \skip_horizontal:N -\arraycolsep }
1559           \@@_use_arraybox_with_notes_c:
1560           \bool_if:NTF \l_@@_NiceTabular_bool
1561           { \skip_horizontal:N -\tabcolsep }
1562           { \skip_horizontal:N -\arraycolsep }
1563         }

```

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

```

1564         \skip_vertical:N -\l_tmpb_dim
1565       }

```

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.

```

1566       \tl_if_empty:NF \l_@@_delimiters_color_tl

```

---

<sup>57</sup>A value of `-1` for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

```

1567         { \color { \l_@@_delimiters_color_tl } }
1568         \exp_after:wN \right \g_@@_right_delim_tl
1569         \c_math_toggle_token
1570     }

```

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.

```

1571     \bool_if:NTF \l_@@_delimiters_max_width_bool
1572     {
1573         \@@_put_box_in_flow_bis:nn
1574         \g_@@_left_delim_tl \g_@@_right_delim_tl
1575     }
1576     \@@_put_box_in_flow:
1577 }

```

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. 107).

```

1578     \bool_if:NT \g_@@_last_col_found_bool
1579     {
1580         \skip_horizontal:N \g_@@_width_last_col_dim
1581         \skip_horizontal:N \col@sep
1582     }
1583     \bool_if:NF \l_@@_Matrix_bool
1584     {
1585         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1586         { \@@_error:n { columns-not-used } }
1587     }
1588     \group_begin:
1589     \globaldefs = 1
1590     \@@_msg_redirect_name:nn { columns-not-used } { error }
1591     \group_end:
1592     \@@_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.

```

1593     \egroup
1594     \bool_if:NT \c_@@_footnote_bool \endsavenotes
1595 }

```

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.

```

1596 \cs_new_protected:Npn \@@_construct_preamble:
1597 {

```

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 `expl3`.

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.

```

1598     \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).

```

1599 \bool_if:NF \l_@@_Matrix_bool
1600 {
1601   \@@_newcolumnstype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1602   \@@_newcolumnstype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by `expl3`).

```

1603 \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1604 \@tempswatrue

```

The following line actually does the expansion (it’s has been copied from `array.sty`).

```

1605 \@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`.

```

1606 \int_gzero_new:N \c@jCol
1607 \tl_gclear:N \g_@@_preamble_tl
1608 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1609 {
1610   \tl_gset:Nn \g_@@_preamble_tl
1611   { ! { \skip_horizontal:N \arrayrulewidth } }
1612 }
1613 {
1614   \clist_if_in:NnT \l_@@_vlines_clist 1
1615   {
1616     \tl_gset:Nn \g_@@_preamble_tl
1617     { ! { \skip_horizontal:N \arrayrulewidth } }
1618   }
1619 }

```

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`).

```

1620 \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1621 \int_zero:N \l_tmpa_int

```

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

```

1622 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1623 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1624 }

```

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

```

1625 \bool_if:NT \l_@@_colortbl_like_bool
1626 {
1627   \regex_replace_all:NnN
1628   \c_@@_columncolor_regex
1629   { \c { @@_columncolor_preamble } }
1630   \g_@@_preamble_tl
1631 }

```

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

```

1632 \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}`.

```

1633 \bool_lazy_or:nnT
1634 { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1635 { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1636 { \bool_set_false:N \l_@@_NiceArray_bool }

```

We complete the preamble with the potential “exterior columns”.

```

1637 \int_compare:nNnTF \l_@@_first_col_int = 0
1638 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1639 {
1640   \bool_lazy_all:nT
1641   {
1642     \l_@@_NiceArray_bool
1643     { \bool_not_p:n \l_@@_NiceTabular_bool }
1644     { \tl_if_empty_p:N \l_@@_vlines_clist }
1645     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1646   }
1647   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1648 }
1649 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1650 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1651 {
1652   \bool_lazy_all:nT
1653   {
1654     \l_@@_NiceArray_bool
1655     { \bool_not_p:n \l_@@_NiceTabular_bool }
1656     { \tl_if_empty_p:N \l_@@_vlines_clist }
1657     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1658   }
1659   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1660 }

```

We add a last column to raise a good error message when the user put 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`).

```

1661 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1662 {
1663   \tl_gput_right:Nn \g_@@_preamble_tl
1664   { > { \@@_error_too_much_cols: } 1 }
1665 }
1666 }

```

```

1667 \cs_new_protected:Npn \@@_patch_preamble:n #1
1668 {
1669   \str_case:nnF { #1 }
1670   {
1671     c { \@@_patch_preamble_i:n #1 }
1672     l { \@@_patch_preamble_i:n #1 }
1673     r { \@@_patch_preamble_i:n #1 }
1674     > { \@@_patch_preamble_ii:nn #1 }
1675     ! { \@@_patch_preamble_ii:nn #1 }
1676     @ { \@@_patch_preamble_ii:nn #1 }
1677     | { \@@_patch_preamble_iii:n #1 }
1678     p { \@@_patch_preamble_iv:nnn t #1 }
1679     m { \@@_patch_preamble_iv:nnn c #1 }
1680     b { \@@_patch_preamble_iv:nnn b #1 }
1681     \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
1682     \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1683     \@@_true_c: { \@@_patch_preamble_vi:n #1 }
1684     ( { \@@_patch_preamble_vii:nn #1 }
1685     [ { \@@_patch_preamble_vii:nn #1 }

```

```

1686 \{ { \@@_patch_preamble_vii:nn #1 }
1687 ) { \@@_patch_preamble_viii:nn #1 }
1688 ] { \@@_patch_preamble_viii:nn #1 }
1689 \} { \@@_patch_preamble_viii:nn #1 }
1690 C { \@@_error:nn { old~column~type } #1 }
1691 L { \@@_error:nn { old~column~type } #1 }
1692 R { \@@_error:nn { old~column~type } #1 }
1693 \q_stop { }
1694 }
1695 {
1696 \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1697 { \@@_patch_preamble_xi:n #1 }
1698 {
1699 \str_if_eq:VnTF \l_@@_letter_vlism_tl { #1 }
1700 {
1701 \seq_gput_right:Nx \g_@@_cols_vlism_seq
1702 { \int_eval:n { \c@jCol + 1 } }
1703 \tl_gput_right:Nx \g_@@_preamble_tl
1704 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1705 \@@_patch_preamble:n
1706 }
1707 {
1708 \bool_lazy_and:nnTF
1709 { \str_if_eq_p:nn { : } { #1 } }
1710 \c_@@_arydshln_loaded_bool
1711 {
1712 \tl_gput_right:Nn \g_@@_preamble_tl { : }
1713 \@@_patch_preamble:n
1714 }
1715 { \@@_fatal:nn { unknown~column~type } { #1 } }
1716 }
1717 }
1718 }
1719 }

```

For c, l and r

```

1720 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1721 {
1722 \tl_gput_right:Nn \g_@@_preamble_tl
1723 {
1724 > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } }
1725 #1
1726 < \@@_end_Cell:
1727 }

```

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

```

1728 \int_gincr:N \c@jCol
1729 \@@_patch_preamble_x:n
1730 }

```

For >, ! and @

```

1731 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1732 {
1733 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1734 \@@_patch_preamble:n
1735 }

```

For |

```

1736 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1737 {

```

\l\_tmpa\_int is the number of successive occurrences of |

```

1738 \int_incr:N \l_tmpa_int
1739 \@@_patch_preamble_iii_i:n
1740 }

```

```

1741 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1742 {
1743   \str_if_eq:nnTF { #1 } |
1744   { \@@_patch_preamble_iii:n | }
1745   {
1746     \tl_gput_right:Nx \g_@@_preamble_tl
1747     {
1748       \exp_not:N !
1749       {
1750         \skip_horizontal:n
1751         {
1752           \dim_eval:n
1753           {
1754             \arrayrulewidth * \l_tmpa_int
1755             + \doublerulesep * ( \l_tmpa_int - 1)
1756           }
1757         }
1758       }
1759     }
1760     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1761     { \@@_vline:nn { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } }
1762     \int_zero:N \l_tmpa_int
1763     \@@_patch_preamble:n #1
1764   }
1765 }

```

For p, m and b

```

1766 \cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
1767 {
1768   \tl_gput_right:Nn \g_@@_preamble_tl
1769   {
1770     > {
1771       \@@_Cell:
1772       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
1773       \mode_leave_vertical:
1774       \arraybackslash
1775       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
1776     }
1777     c
1778     < {
1779       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
1780       \end { minipage }
1781       \@@_end_Cell:
1782     }
1783   }

```

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

```

1784   \int_gincr:N \c@jCol
1785   \@@_patch_preamble_x:n
1786 }

```

For w and W

```

1787 \cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
1788 {
1789   \tl_gput_right:Nn \g_@@_preamble_tl
1790   {
1791     > {
1792       \hbox_set:Nw \l_@@_cell_box
1793       \@@_Cell:
1794       \tl_set:Nn \l_@@_cell_type_tl { #3 }
1795     }
1796     c
1797     < {
1798       \@@_end_Cell:

```

```

1799         #1
1800         \hbox_set_end:
1801         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1802         \@@_adjust_size_box:
1803         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
1804     }
1805 }

```

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

```

1806     \int_gincr:N \c@jCol
1807     \@@_patch_preamble_x:n
1808 }

```

For \@@\_true\_c: which will appear in our redefinition of the columns of type S (of siunitx).

```

1809 \cs_new_protected:Npn \@@_patch_preamble_vi:n #1
1810 {
1811     \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

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

```

1812     \int_gincr:N \c@jCol
1813     \@@_patch_preamble_x:n
1814 }

```

For (, [ and \{.

```

1815 \cs_new_protected:Npn \@@_patch_preamble_vii:nn #1 #2
1816 {
1817     \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.

```

1818     \int_compare:nNnTF \c@jCol = \c_zero_int
1819     {
1820         \str_if_eq:VnTF \g_@@_left_delim_tl { . }
1821         {

```

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

```

1822             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1823             \tl_gset:Nn \g_@@_right_delim_tl { . }
1824             \@@_patch_preamble:n #2
1825         }
1826     {
1827         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1828         \tl_gput_right:Nx \g_@@_internal_code_after_tl
1829         { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
1830         \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
1831         {
1832             \@@_error:nn { delimiter~after~opening } { #2 }
1833             \@@_patch_preamble:n
1834         }
1835         { \@@_patch_preamble:n #2 }
1836     }
1837 }
1838 {
1839     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1840     { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
1841     \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
1842     {
1843         \@@_error:nn { delimiter~after~opening } { #2 }
1844         \@@_patch_preamble:n
1845     }
1846     { \@@_patch_preamble:n #2 }
1847 }
1848 }

```

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}`).

```

1849 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
1850 {
1851   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter-with-small } }
1852   \tl_if_in:nnTF { ) ] \} } { #2 }
1853   { \@@_patch_preamble_viii_i:nnn #1 #2 }
1854   {
1855     \tl_if_eq:nnTF { \q_stop } { #2 }
1856     {
1857       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
1858       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
1859       {
1860         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1861         \tl_gput_right:Nx \g_@@_internal_code_after_tl
1862           { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1863         \@@_patch_preamble:n #2
1864       }
1865     }
1866     {
1867       \tl_if_in:nnT { ( [ \{ } { #2 }
1868       { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
1869       \tl_gput_right:Nx \g_@@_internal_code_after_tl
1870         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1871       \@@_patch_preamble:n #2
1872     }
1873   }
1874 }

1875 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nnn #1 #2 #3
1876 {
1877   \tl_if_eq:nnTF { \q_stop } { #3 }
1878   {
1879     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
1880     {
1881       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1882       \tl_gput_right:Nx \g_@@_internal_code_after_tl
1883         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1884       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1885     }
1886     {
1887       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1888       \tl_gput_right:Nx \g_@@_internal_code_after_tl
1889         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1890       \@@_error:nn { double-closing-delimiter } { #2 }
1891     }
1892   }
1893   {
1894     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1895       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1896     \@@_error:nn { double-closing-delimiter } { #2 }
1897     \@@_patch_preamble:n #3
1898   }
1899 }

1900 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
1901 {
1902   \tl_gput_right:Nn \g_@@_preamble_tl
1903     { ! { \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`.



```

1904 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1905 { \@@_vdottedline:n { \int_use:N \c@jCol } }
1906 \@@_patch_preamble:n
1907 }

```

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.

```

1908 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
1909 {
1910   \str_if_eq:nnTF { #1 } { < }
1911   \@@_patch_preamble_ix:n
1912   {
1913     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1914     {
1915       \tl_gput_right:Nn \g_@@_preamble_tl
1916       { ! { \skip_horizontal:N \arrayrulewidth } }
1917     }
1918     {
1919       \exp_args:NNx
1920       \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
1921       {
1922         \tl_gput_right:Nn \g_@@_preamble_tl
1923         { ! { \skip_horizontal:N \arrayrulewidth } }
1924       }
1925     }
1926     \@@_patch_preamble:n { #1 }
1927   }
1928 }
1929 \cs_new_protected:Npn \@@_patch_preamble_ix:n #1
1930 {
1931   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
1932   \@@_patch_preamble_x:n
1933 }

```

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`).

```

1934 \cs_new_protected:Npn \@@_put_box_in_flow:
1935 {
1936   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
1937   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
1938   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
1939   { \box_use_drop:N \l_tmpa_box }
1940   \@@_put_box_in_flow_i:
1941 }

```

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).

```

1942 \cs_new_protected:Npn \@@_put_box_in_flow_i:
1943 {
1944   \pgfpicture
1945   \@@_qpoint:n { row - 1 }
1946   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1947   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
1948   \dim_gadd:Nn \g_tmpa_dim \pgf@y
1949   \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).

```

1950   \str_if_in:NnTF \l_@@_baseline_tl { line- }
1951   {

```

```

1952 \int_set:Nn \l_tmpa_int
1953 {
1954   \str_range:Nnn
1955     \l_@@_baseline_tl
1956     6
1957   { \tl_count:V \l_@@_baseline_tl }
1958 }
1959 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
1960 }
1961 {
1962   \str_case:VnF \l_@@_baseline_tl
1963   {
1964     { t } { \int_set:Nn \l_tmpa_int 1 }
1965     { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
1966   }
1967   { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
1968   \bool_lazy_or:nnT
1969     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
1970     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
1971   {
1972     \@@_error:n { bad~value~for~baseline }
1973     \int_set:Nn \l_tmpa_int 1
1974   }
1975   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

1976 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
1977 }
1978 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

1979 \endpgfpicture
1980 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
1981 \box_use_drop:N \l_tmpa_box
1982 }

```

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).

```

1983 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
1984 {

```

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

```

1985 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
1986 \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.

```

1987 \@@_create_extra_nodes:
1988 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
1989 \bool_lazy_or:nnT
1990   { \int_compare_p:nNn \c@tabularnote > 0 }
1991   { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
1992   \@@_insert_tabularnotes:
1993 \end { minipage }
1994 }
1995 \cs_new_protected:Npn \@@_insert_tabularnotes:
1996 {
1997   \skip_vertical:N 0.65ex

```

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

```

1998 \group_begin:
1999 \l_@@_notes_code_before_tl
2000 \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.

```

2001 \int_compare:nNnT \c@tabularnote > 0
2002 {
2003   \bool_if:NTF \l_@@_notes_para_bool
2004   {
2005     \begin { tabularnotes* }
2006     \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2007     \end { tabularnotes* }

```

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

```

2008   \par
2009 }
2010 {
2011   \tabularnotes
2012   \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2013   \endtabularnotes
2014 }
2015 }
2016 \unskip
2017 \group_end:
2018 \bool_if:NT \l_@@_notes_bottomrule_bool
2019 {
2020   \bool_if:NTF \c_@@_booktabs_loaded_bool
2021   {

```

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

```

2022   \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2023   { \CT@arc@ \hrule height \heavyrulewidth }
2024   }
2025   { \@_error:n { bottomrule~without~booktabs } }
2026   }
2027   \l_@@_notes_code_after_tl
2028   \seq_gclear:N \g_@@_tabularnotes_seq
2029   \int_gzero:N \c@tabularnote
2030 }

```

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`.

```

2031 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2032 {
2033   \pgfpicture
2034   \@@_qpoint:n { row - 1 }
2035   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2036   \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2037   \dim_gsub:Nn \g_tmpa_dim \pgf@y
2038   \endpgfpicture
2039   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2040   \int_compare:nNnT \l_@@_first_row_int = 0
2041   {
2042     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2043     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2044   }
2045   \box_move_up:nn \g_tmpa_dim { \hbox { \@_use_arraybox_with_notes_c: } }
2046 }

```

Now, the general case.

```
2047 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2048 {
```

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

```
2049 \tl_if_eq:NnT \l_@@_baseline_tl { t }
2050 { \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`.

```
2051 \pgfpicture
2052 \@@_qpoint:n { row - 1 }
2053 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2054 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2055 {
2056   \int_set:Nn \l_tmpa_int
2057   {
2058     \str_range:Nnn
2059     \l_@@_baseline_tl
2060     6
2061     { \tl_count:V \l_@@_baseline_tl }
2062   }
2063   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2064 }
2065 {
2066   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2067   \bool_lazy_or:nnT
2068   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2069   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2070   {
2071     \@@_error:n { bad-value-for-baseline }
2072     \int_set:Nn \l_tmpa_int 1
2073   }
2074   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2075 }
2076 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2077 \endpgfpicture
2078 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2079 \int_compare:nNnT \l_@@_first_row_int = 0
2080 {
2081   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2082   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2083 }
2084 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2085 }
```

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.

```
2086 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2087 {
```

We will compute the real width of both delimiters used.

```
2088 \dim_zero_new:N \l_@@_real_left_delim_dim
2089 \dim_zero_new:N \l_@@_real_right_delim_dim
2090 \hbox_set:Nn \l_tmpb_box
2091 {
2092   \c_math_toggle_token
2093   \left #1
2094   \vcenter
2095   {
2096     \vbox_to_ht:nn
2097     { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2098     { }
```

```

2099     }
2100     \right .
2101     \c_math_toggle_token
2102   }
2103   \dim_set:Nn \l_@@_real_left_delim_dim
2104   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2105   \hbox_set:Nn \l_tmpb_box
2106   {
2107     \c_math_toggle_token
2108     \left .
2109     \vbox_to_ht:nn
2110     { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2111     { }
2112     \right #2
2113     \c_math_toggle_token
2114   }
2115   \dim_set:Nn \l_@@_real_right_delim_dim
2116   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2117   \skip_horizontal:N \l_@@_left_delim_dim
2118   \skip_horizontal:N -\l_@@_real_left_delim_dim
2119   \@@_put_box_in_flow:
2120   \skip_horizontal:N \l_@@_right_delim_dim
2121   \skip_horizontal:N -\l_@@_real_right_delim_dim
2122 }

```

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).

```

2123 \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).

```

2124 {
2125   \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`).

```

2126   { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2127 }
2128 {
2129   \@@_create_col_nodes:
2130   \endarray
2131 }

```

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`).

```

2132 \NewDocumentEnvironment { @@-light-syntax } { b }
2133 {

```

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`.

```

2134   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2135   \tl_map_inline:nn { #1 }
2136   {
2137     \str_if_eq:nnT { ##1 } { & }
2138     { \@@_fatal:n { ampersand-in-light-syntax } }

```

```

2139     \str_if_eq:nnT { ##1 } { \\\ }
2140     { \@@_fatal:n { double-backslash-in~light-syntax } }
2141 }

```

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`.

```

2142     \@@_light_syntax_i #1 \CodeAfter \q_stop
2143 }

```

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`.

```

2144 { }
2145 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2146 {
2147     \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).

```

2148     \seq_gclear_new:N \g_@@_rows_seq
2149     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2150     \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.

```

2151     \int_compare:nnnT \l_@@_last_row_int = { -1 }
2152     { \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`).

```

2153     \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.

```

2154     \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2155     \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
2156     \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2157     \@@_create_col_nodes:
2158     \endarray
2159 }
2160 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2161 { \tl_if_empty:nF { #1 } { \\\ \@@_line_with_light_syntax_i:n { #1 } } }
2162 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2163 {
2164     \seq_gclear_new:N \g_@@_cells_seq
2165     \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2166     \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2167     \l_tmpa_tl
2168     \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2169 }

```

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).

```

2170 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2171 {
2172     \str_if_eq:VnT \g_@@_name_env_str { #2 }
2173     { \@@_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.

```

2174     \end { #2 }
2175 }

```

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 specify the width of the columns).

```

2176 \cs_new:Npn \@@_create_col_nodes:
2177 {
2178   \crrc
2179   \int_compare:nNnT \l_@@_first_col_int = 0
2180   {
2181     \omit
2182     \hbox_overlap_left:n
2183     {
2184       \bool_if:NT \l_@@_code_before_bool
2185       { \pgfsys@markposition { \@@_env: - col - 0 } }
2186       \pgfpicture
2187       \pgfrememberpicturerepositiononpagetrue
2188       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
2189       \str_if_empty:NF \l_@@_name_str
2190       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2191       \endpgfpicture
2192       \skip_horizontal:N 2\col@sep
2193       \skip_horizontal:N \g_@@_width_first_col_dim
2194     }
2195     &
2196   }
2197   \omit

```

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

```

2198   \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`).

```

2199   \int_compare:nNnTF \l_@@_first_col_int = 0
2200   {
2201     \bool_if:NT \l_@@_code_before_bool
2202     {
2203       \hbox
2204       {
2205         \skip_horizontal:N -0.5\arrayrulewidth
2206         \pgfsys@markposition { \@@_env: - col - 1 }
2207         \skip_horizontal:N 0.5\arrayrulewidth
2208       }
2209     }
2210     \pgfpicture
2211     \pgfrememberpicturerepositiononpagetrue
2212     \pgfcoordinate { \@@_env: - col - 1 }
2213     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
2214     \str_if_empty:NF \l_@@_name_str
2215     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2216     \endpgfpicture
2217   }
2218   {
2219     \bool_if:NT \l_@@_code_before_bool
2220     {
2221       \hbox
2222       {
2223         \skip_horizontal:N 0.5\arrayrulewidth
2224         \pgfsys@markposition { \@@_env: - col - 1 }
2225         \skip_horizontal:N -0.5\arrayrulewidth
2226       }
2227     }
2228     \pgfpicture
2229     \pgfrememberpicturerepositiononpagetrue
2230     \pgfcoordinate { \@@_env: - col - 1 }
2231     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }

```

```

2232     \str_if_empty:NF \l_@@_name_str
2233     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2234     \endpgfpicture
2235 }

```

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.

```

2236     \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2237     \bool_if:NF \l_@@_auto_columns_width_bool
2238     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2239     {
2240         \bool_lazy_and:nnTF
2241         \l_@@_auto_columns_width_bool
2242         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2243         { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2244         { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2245         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2246     }
2247     \skip_horizontal:N \g_tmpa_skip
2248     \hbox
2249     {
2250         \bool_if:NT \l_@@_code_before_bool
2251         {
2252             \hbox
2253             {
2254                 \skip_horizontal:N -0.5\arrayrulewidth
2255                 \pgfsys@markposition { \@@_env: - col - 2 }
2256                 \skip_horizontal:N 0.5\arrayrulewidth
2257             }
2258         }
2259         \pgfpicture
2260         \pgfrememberpicturepositiononpagetrue
2261         \pgfcoordinate { \@@_env: - col - 2 }
2262         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2263         \str_if_empty:NF \l_@@_name_str
2264         { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2265         \endpgfpicture
2266     }

```

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.

```

2267     \int_gset:Nn \g_tmpa_int 1
2268     \bool_if:NTF \g_@@_last_col_found_bool
2269     { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
2270     { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
2271     {
2272         &
2273         \omit
2274         \int_gincr:N \g_tmpa_int

```

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

```

2275         \skip_horizontal:N \g_tmpa_skip
2276         \bool_if:NT \l_@@_code_before_bool
2277         {
2278             \hbox
2279             {
2280                 \skip_horizontal:N -0.5\arrayrulewidth
2281                 \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2282                 \skip_horizontal:N 0.5\arrayrulewidth
2283             }

```



2284 }

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

```

2285 \pgfpicture
2286 \pgfrememberpicturepositiononpagetrue
2287 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2288 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2289 \str_if_empty:NF \l_@@_name_str
2290 {
2291 \pgfnodealias
2292 { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2293 { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2294 }
2295 \endpgfpicture
2296 }
2297 \bool_if:NT \g_@@_last_col_found_bool
2298 {
2299 \hbox_overlap_right:n
2300 {
2301 % \skip_horizontal:N \col@sep
2302 \skip_horizontal:N \g_@@_width_last_col_dim
2303 \bool_if:NT \l_@@_code_before_bool
2304 {
2305 \pgfsys@markposition
2306 { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2307 }
2308 \pgfpicture
2309 \pgfrememberpicturepositiononpagetrue
2310 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2311 \pgfpointorigin
2312 \str_if_empty:NF \l_@@_name_str
2313 {
2314 \pgfnodealias
2315 { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2316 { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2317 }
2318 \endpgfpicture
2319 }
2320 }
2321 \cr
2322 }

```

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

```

2323 \tl_const:Nn \c_@@_preamble_first_col_tl
2324 {
2325 >
2326 {

```

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

```

2327 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
2328 \bool_gset_true:N \g_@@_after_col_zero_bool
2329 \@@_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.

```

2330 \hbox_set:Nw \l_@@_cell_box
2331 \@@_math_toggle_token:
2332 \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”.

```

2333 \bool_lazy_and:nnT
2334 { \int_compare_p:nNn \c@iRow > 0 }

```

```

2335     {
2336         \bool_lazy_or_p:nn
2337         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2338         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2339     }
2340     {
2341         \l_@@_code_for_first_col_tl
2342         \xglobal \colorlet { nicematrix-first-col } { . }
2343     }
2344 }

```

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`.

```

2345     l
2346     <
2347     {
2348         \@@_math_toggle_token:
2349         \hbox_set_end:
2350         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2351         \@@_adjust_size_box:
2352         \@@_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.

```

2353     \dim_gset:Nn \g_@@_width_first_col_dim
2354     { \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.

```

2355     \hbox_overlap_left:n
2356     {
2357         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2358         \@@_node_for_cell:
2359         { \box_use_drop:N \l_@@_cell_box }
2360         \skip_horizontal:N \l_@@_left_delim_dim
2361         \skip_horizontal:N \l_@@_left_margin_dim
2362         \skip_horizontal:N \l_@@_extra_left_margin_dim
2363     }
2364     \bool_gset_false:N \g_@@_empty_cell_bool
2365     \skip_horizontal:N -2\col@sep
2366 }
2367 }

```

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

```

2368 \tl_const:Nn \c_@@_preamble_last_col_tl
2369 {
2370     >
2371     {

```

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

```

2372     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n

```

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

```

2373     \bool_gset_true:N \g_@@_last_col_found_bool
2374     \int_gincr:N \c@jCol
2375     \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.

```

2376     \hbox_set:Nw \l_@@_cell_box
2377     \@@_math_toggle_token:
2378     \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”.

```

2379     \int_compare:nNnT \c@iRow > 0
2380     {
2381         \bool_lazy_or:nnT
2382         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2383         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2384         {
2385             \l_@@_code_for_last_col_tl
2386             \xglobal \colorlet { nicematrix-last-col } { . }
2387         }
2388     }
2389 }
2390
2391 <
2392 {
2393     \@@_math_toggle_token:
2394     \hbox_set_end:
2395     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2396     \@@_adjust_size_box:
2397     \@@_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.

```

2398     \dim_gset:Nn \g_@@_width_last_col_dim
2399     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2400     \skip_horizontal:N -2\col@sep

```

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

```

2401     \hbox_overlap_right:n
2402     {
2403         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2404         {
2405             \skip_horizontal:N \l_@@_right_delim_dim
2406             \skip_horizontal:N \l_@@_right_margin_dim
2407             \skip_horizontal:N \l_@@_extra_right_margin_dim
2408             \@@_node_for_cell:
2409         }
2410     }
2411     \bool_gset_false:N \g_@@_empty_cell_bool
2412 }
2413 }

```

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.

```

2414 \NewDocumentEnvironment { NiceArray } { }
2415 {
2416     \bool_set_true:N \l_@@_NiceArray_bool
2417     \str_if_empty:NT \g_@@_name_env_str
2418     { \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).

```

2419     \NiceArrayWithDelims . .
2420 }
2421 { \endNiceArrayWithDelims }

```

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

```

2422 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2423 {

```

```

2424 \NewDocumentEnvironment { #1 NiceArray } { }
2425 {
2426   \str_if_empty:NT \g_@@_name_env_str
2427   { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2428   \@@_test_if_math_mode:
2429   \NiceArrayWithDelims #2 #3
2430 }
2431 { \endNiceArrayWithDelims }
2432 }
2433 \@@_def_env:nnn p ( )
2434 \@@_def_env:nnn b [ ]
2435 \@@_def_env:nnn B \{ \}
2436 \@@_def_env:nnn v | |
2437 \@@_def_env:nnn V \| \|

```

## The environment {NiceMatrix} and its variants

```

2438 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2439 {
2440   \bool_set_true:N \l_@@_Matrix_bool
2441   \use:c { #1 NiceArray }
2442   {
2443     *
2444     {
2445       \int_compare:nNnTF \l_@@_last_col_int < 0
2446       \c@MaxMatrixCols
2447       { \@@_pred:n \l_@@_last_col_int }
2448     }
2449     { > \@@_Cell: #2 < \@@_end_Cell: }
2450   }
2451 }
2452 \clist_map_inline:nn { { } , p , b , B , v , V }
2453 {
2454   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2455   {
2456     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2457     \tl_set:Nn \l_@@_type_of_col_tl c
2458     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2459     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2460   }
2461   { \use:c { end #1 NiceArray } }
2462 }

```

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

```

2463 \cs_new_protected:Npn \@@_NotEmpty:
2464 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

## The environments {NiceTabular} and {NiceTabular\*}

```

2465 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2466 {
2467   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2468   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2469   \bool_set_true:N \l_@@_NiceTabular_bool
2470   \NiceArray { #2 }
2471 }
2472 { \endNiceArray }
2473 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }

```

```

2474 {
2475   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2476   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
2477   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2478   \bool_set_true:N \l_@@_NiceTabular_bool
2479   \NiceArray { #3 }
2480 }
2481 { \endNiceArray }

```

## After the construction of the array

```

2482 \cs_new_protected:Npn \@@_after_array:
2483 {
2484   \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.

```

2485   \bool_if:NT \g_@@_last_col_found_bool
2486   { \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 fix the real value of `\l_@@_last_col_int`.

```

2487   \bool_if:NT \l_@@_last_col_without_value_bool
2488   {
2489     \dim_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int
2490     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2491     \iow_shipout:Nx \@mainaux
2492     {
2493       \cs_gset:cpn { @@_last_col_ \int_use:N \g_@@_env_int }
2494       { \int_use:N \g_@@_col_total_int }
2495     }
2496     \str_if_empty:NF \l_@@_name_str
2497     {
2498       \iow_shipout:Nx \@mainaux
2499       {
2500         \cs_gset:cpn { @@_last_col_ \l_@@_name_str }
2501         { \int_use:N \g_@@_col_total_int }
2502       }
2503     }
2504     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2505   }

```

It's also time to give to `\l_@@_last_row_int` its real value. But, if the user had used the option `last-row` without value, we write in the aux file the number of that last row for the next run.

```

2506   \bool_if:NT \l_@@_last_row_without_value_bool
2507   {
2508     \dim_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int

```

If the option `light-syntax` is used, we have nothing to write since, in this case, the number of rows is directly determined.

```

2509   \bool_if:NF \l_@@_light_syntax_bool
2510   {
2511     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2512     \iow_shipout:Nx \@mainaux
2513     {
2514       \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int }
2515       { \int_use:N \g_@@_row_total_int }
2516     }

```

If the environment has a name, we also write a value based on the name because it's more reliable than a value based on the number of the environment.

```

2517         \str_if_empty:NF \l_@@_name_str
2518         {
2519             \iow_shipout:Nx \@mainaux
2520             {
2521                 \cs_gset:cpn { @@_last_row_ \l_@@_name_str }
2522                 { \int_use:N \g_@@_row_total_int }
2523             }
2524         }
2525         \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2526     }
2527 }

```

If the key `code-before` is used, we have to write on the aux file the actual size of the array and other informations.

```

2528     \bool_if:NT \l_@@_code_before_bool
2529     {
2530         \iow_now:Nn \@mainaux \ExplSyntaxOn
2531         \iow_now:Nx \@mainaux
2532         { \seq_clear_new:c { @@_size _ \int_use:N \g_@@_env_int _ seq } }
2533         \iow_now:Nx \@mainaux
2534         {
2535             \seq_gset_from_clist:cn { @@_size _ \int_use:N \g_@@_env_int _ seq }
2536             {
2537                 \int_use:N \l_@@_first_row_int ,
2538                 \int_use:N \g_@@_row_total_int ,
2539                 \int_use:N \l_@@_first_col_int ,

```

If the user has used a key `last-row` in an environment with preamble (like `{pNiceArray}`) and that that last row has not been found, we have to increment the value because it will be decreased when used in the `code-before`.

```

2540         \bool_lazy_and:nnTF
2541         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
2542         { \bool_not_p:n \g_@@_last_col_found_bool }
2543         \@@_succ:n
2544         \int_use:N
2545         \g_@@_col_total_int
2546     }

```

We write also the potential content of `\g_@@_pos_of_blocks_seq` (it will be useful if the command `\rowcolors` is used with the key `respect-blocks`).

```

2547         \seq_gset_from_clist:cn
2548         { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
2549         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
2550     }
2551     \iow_now:Nn \@mainaux \ExplSyntaxOff
2552 }

```

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

```

2553     \@@_create_diag_nodes:

```

By default, the diagonal lines will be parallelized<sup>58</sup>. 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.

```

2554     \bool_if:NT \l_@@_parallelize_diags_bool
2555     {
2556         \int_gzero_new:N \g_@@_ddots_int
2557         \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the  $\Delta_x$  and  $\Delta_y$  of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the  $\Delta_x$  and  $\Delta_y$  of the first `\Iddots` diagonal.

```

2558         \dim_gzero_new:N \g_@@_delta_x_one_dim

```

---

<sup>58</sup>It's possible to use the option `parallelize-diags` to disable this parallelization.

```

2559     \dim_gzero_new:N \g_@@_delta_y_one_dim
2560     \dim_gzero_new:N \g_@@_delta_x_two_dim
2561     \dim_gzero_new:N \g_@@_delta_y_two_dim
2562   }
2563   \int_zero_new:N \l_@@_initial_i_int
2564   \int_zero_new:N \l_@@_initial_j_int
2565   \int_zero_new:N \l_@@_final_i_int
2566   \int_zero_new:N \l_@@_final_j_int
2567   \bool_set_false:N \l_@@_initial_open_bool
2568   \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_radius_dim` and `\l_@@_inter_dots_dim` (used to draw the dotted lines created by `\hdottedline` and `\vdotteline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.

```

2569   \bool_if:NT \l_@@_small_bool
2570   {
2571     \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
2572     \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }

```

The dimension `\l_@@_xdots_shorten_dim` corresponds to the option `xdots/shorten` available to the user. That's why we give a new value according to the current value, and not an absolute value.

```

2573     \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
2574   }

```

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

```

2575   \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won't do anything. The corners are computed in `\l_@@_corners_cells_seq` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

2576   \@@_compute_corners:

```

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

```

2577   \@@_adjust_pos_of_blocks_seq:

```

The following code is only for efficiency. We determine whether the potential horizontal and vertical rules are “complete”, that is to say drawn in the whole array. We are sure that all the rules will be complete when there is no block, no virtual block (determined by a command such as `\Cdots`, `\Vdots`, etc.) and no corners. In that case, we switch to a shortcut version of `\@@_vline_i:nn` and `\@@_hline:nn`.

```

2578   \bool_lazy_all:nT
2579   {
2580     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
2581     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
2582     { \seq_if_empty_p:N \l_@@_corners_cells_seq }
2583   }
2584   {
2585     \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn
2586     \cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn
2587   }
2588   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
2589   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
2590   \cs_set_eq:NN \SubMatrix \@@_SubMatrix

```

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

```

2591   \bool_if:NT \c_@@_tikz_loaded_bool
2592   {
2593     \tikzset
2594     {
2595       every~picture / .style =

```

```

2596         {
2597             overlay ,
2598             remember~picture ,
2599             name~prefix = \@@_env: -
2600         }
2601     }
2602 }
2603 \cs_set_eq:NN \line \@@_line
2604 \g_@@_internal_code_after_tl
2605 \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.

```

2606 \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).

```

2607 \seq_gclear:N \g_@@_submatrix_names_seq

```

We compose the `code-after` in math mode in order to nullify the spaces put by the user between instructions in the `code-after`.

```

2608 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token

```

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:`.

```

2609 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
2610 \scan_stop:
2611 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
2612 \tl_gclear:N \g_nicematrix_code_after_tl
2613 \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.

```

2614 \tl_if_empty:NF \g_nicematrix_code_before_tl
2615 {

```

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`.

```

2616 \cs_set_protected:Npn \rectanglecolor { }
2617 \cs_set_protected:Npn \columncolor { }
2618 \iow_now:Nn \@mainaux \ExplSyntaxOn
2619 \iow_now:Nx \@mainaux
2620 {
2621     \tl_gset:cn
2622     { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
2623     { \exp_not:V \g_nicematrix_code_before_tl }
2624 }
2625 \iow_now:Nn \@mainaux \ExplSyntaxOff
2626 \bool_set_true:N \l_@@_code_before_bool
2627 }
2628 % \bool_if:NT \l_@@_code_before_bool \@@_write_aux_for_cell_nodes:

2629 \str_gclear:N \g_@@_name_env_str
2630 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>59</sup>. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able

---

<sup>59</sup>e.g. `\color[rgb]{0.5,0.5,0}`



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`.

```
2631 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
2632 }
```

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`.

```
2633 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
2634 { \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).

```
2635 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
2636 {
2637   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
2638   { \@@_adjust_pos_of_blocks_seq_i:nnnn ##1 }
2639 }
```

The following command must *not* be protected.

```
2640 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnn #1 #2 #3 #4
2641 {
2642   { #1 }
2643   { #2 }
2644   {
2645     \int_compare:nNnTF { #3 } > { 99 }
2646     { \int_use:N \c@iRow }
2647     { #3 }
2648   }
2649   {
2650     \int_compare:nNnTF { #4 } > { 99 }
2651     { \int_use:N \c@jCol }
2652     { #4 }
2653   }
2654 }
```

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).

```
2655 \AtBeginDocument
2656 {
2657   \cs_new_protected:Npx \@@_draw_dotted_lines:
2658   {
2659     \c_@@_pgfortikzpicture_tl
2660     \@@_draw_dotted_lines_i:
2661     \c_@@_endpgfortikzpicture_tl
2662   }
2663 }
```

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

```
2664 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
2665 {
2666   \pgfrememberpicturepositiononpagetrue
2667   \pgf@relevantforpicturesizefalse
2668   \g_@@_HVdotsfor_lines_tl
2669   \g_@@_Vdots_lines_tl
2670   \g_@@_Ddots_lines_tl
2671 }
```

```

2671 \g_@@_Iddots_lines_tl
2672 \g_@@_Cdots_lines_tl
2673 \g_@@_Ldots_lines_tl
2674 }

2675 \cs_new_protected:Npn \@@_restore_iRow_jCol:
2676 {
2677   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
2678   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
2679 }

```

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

```

2680 \pgfdeclareshape { @@_diag_node }
2681 {
2682   \savedanchor { \five }
2683   {
2684     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
2685     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
2686   }
2687   \anchor { 5 } { \five }
2688   \anchor { center } { \pgfpointorigin }
2689 }

2690 \cs_new_protected:Npn \@@_write_aux_for_cell_nodes:
2691 {
2692   \pgfpicture
2693   \pgfrememberpicturerepositiononpagetrue
2694   \pgf@relevantforpicturesizefalse
2695   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2696   {
2697     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2698     {
2699       \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
2700       {
2701         \pgfscope
2702         \pgftransformshift
2703         { \pgfpointanchor { \@@_env: - ##1 - #####1 } { north-west } }
2704         \pgfnode
2705         { rectangle }
2706         { center }
2707         {
2708           \hbox
2709           { \pgfsys@markposition { \@@_env: - ##1 - #####1 - NW } }
2710         }
2711         { }
2712         { }
2713       \endpgfscope
2714       \pgfscope
2715       \pgftransformshift
2716       { \pgfpointanchor { \@@_env: - ##1 - #####1 } { south-east } }
2717       \pgfnode
2718       { rectangle }
2719       { center }
2720       {
2721         \hbox
2722         { \pgfsys@markposition { \@@_env: - ##1 - #####1 - SE } }
2723       }
2724       { }
2725       { }
2726     \endpgfscope
2727   }

```

```

2728     }
2729   }
2730   \endpgfpicture
2731   \@@_create_extra_nodes:
2732 }
2733 % \end{macrocode}
2734 %
2735 % \bigskip
2736 % The following command creates the diagonal nodes (in fact, if the matrix is
2737 % not a square matrix, not all the nodes are on the diagonal).
2738 % \begin{macrocode}
2739 \cs_new_protected:Npn \@@_create_diag_nodes:
2740 {
2741   \pgfpicture
2742   \pgfrememberpicturepositiononpagetrue
2743   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
2744   {
2745     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
2746     \dim_set_eq:NN \l_tmpa_dim \pgf@x
2747     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
2748     \dim_set_eq:NN \l_tmpb_dim \pgf@y
2749     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
2750     \dim_set_eq:NN \l_tmpc_dim \pgf@x
2751     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
2752     \dim_set_eq:NN \l_tmpd_dim \pgf@y
2753     \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.

```

2754     \dim_set:Nn \l_tmpa_dim { ( \l_tmpc_dim - \l_tmpa_dim ) / 2 }
2755     \dim_set:Nn \l_tmpb_dim { ( \l_tmpd_dim - \l_tmpb_dim ) / 2 }
2756     \pgfnode { @â_diag_node } { center } { } { \@@_env: - ##1 } { }
2757   }

```

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

```

2758   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
2759   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
2760   \dim_set_eq:NN \l_tmpa_dim \pgf@y
2761   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
2762   \pgfcoordinate
2763   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
2764   \pgfnodealias
2765   { \@@_env: - last }
2766   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
2767   \endpgfpicture
2768 }

```

## 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 `\@@_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.

```
2769 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
2770 {
```

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

```
2771 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
2772 \int_set:Nn \l_@@_initial_i_int { #1 }
2773 \int_set:Nn \l_@@_initial_j_int { #2 }
2774 \int_set:Nn \l_@@_final_i_int { #1 }
2775 \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.

```
2776 \bool_set_false:N \l_@@_stop_loop_bool
2777 \bool_do_until:Nn \l_@@_stop_loop_bool
2778 {
2779   \int_add:Nn \l_@@_final_i_int { #3 }
2780   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
2781   \bool_set_false:N \l_@@_final_open_bool
2782   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
2783   {
2784     \int_compare:nNnTF { #3 } = 1
2785     { \bool_set_true:N \l_@@_final_open_bool }
2786     {
2787       \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
2788       { \bool_set_true:N \l_@@_final_open_bool }
2789     }
2790   }
2791 {
2792   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
2793   {
2794     \int_compare:nNnTF { #4 } = { -1 }
2795     { \bool_set_true:N \l_@@_final_open_bool }
2796   }
2797 {
2798   \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
2799   {
2800     \int_compare:nNnTF { #4 } = 1
2801     { \bool_set_true:N \l_@@_final_open_bool }
2802   }
2803 }
2804 }
2805 \bool_if:NNTF \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.

```
2806 {
```

We do a step backwards.

```

2807         \int_sub:Nn \l_@@_final_i_int { #3 }
2808         \int_sub:Nn \l_@@_final_j_int { #4 }
2809         \bool_set_true:N \l_@@_stop_loop_bool
2810     }

```

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`.

```

2811     {
2812         \cs_if_exist:cTF
2813         {
2814             @@ _ dotted _
2815             \int_use:N \l_@@_final_i_int -
2816             \int_use:N \l_@@_final_j_int
2817         }
2818         {
2819             \int_sub:Nn \l_@@_final_i_int { #3 }
2820             \int_sub:Nn \l_@@_final_j_int { #4 }
2821             \bool_set_true:N \l_@@_final_open_bool
2822             \bool_set_true:N \l_@@_stop_loop_bool
2823         }
2824         {
2825             \cs_if_exist:cTF
2826             {
2827                 pgf @ sh @ ns @ \@@_env:
2828                 - \int_use:N \l_@@_final_i_int
2829                 - \int_use:N \l_@@_final_j_int
2830             }
2831             { \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.

```

2832         {
2833             \cs_set:cpn
2834             {
2835                 @@ _ dotted _
2836                 \int_use:N \l_@@_final_i_int -
2837                 \int_use:N \l_@@_final_j_int
2838             }
2839             { }
2840         }
2841     }
2842 }
2843 }

```

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

```

2844     \bool_set_false:N \l_@@_stop_loop_bool
2845     \bool_do_until:Nn \l_@@_stop_loop_bool
2846     {
2847         \int_sub:Nn \l_@@_initial_i_int { #3 }
2848         \int_sub:Nn \l_@@_initial_j_int { #4 }
2849         \bool_set_false:N \l_@@_initial_open_bool
2850         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
2851         {
2852             \int_compare:nNnTF { #3 } = 1
2853             { \bool_set_true:N \l_@@_initial_open_bool }
2854             {
2855                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }

```

```

2856         { \bool_set_true:N \l_@@_initial_open_bool }
2857     }
2858 }
2859 {
2860     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
2861     {
2862         \int_compare:nNnT { #4 } = 1
2863         { \bool_set_true:N \l_@@_initial_open_bool }
2864     }
2865     {
2866         \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
2867         {
2868             \int_compare:nNnT { #4 } = { -1 }
2869             { \bool_set_true:N \l_@@_initial_open_bool }
2870         }
2871     }
2872 }
2873 \bool_if:NTF \l_@@_initial_open_bool
2874 {
2875     \int_add:Nn \l_@@_initial_i_int { #3 }
2876     \int_add:Nn \l_@@_initial_j_int { #4 }
2877     \bool_set_true:N \l_@@_stop_loop_bool
2878 }
2879 {
2880     \cs_if_exist:cTF
2881     {
2882         @@ _ dotted _
2883         \int_use:N \l_@@_initial_i_int -
2884         \int_use:N \l_@@_initial_j_int
2885     }
2886     {
2887         \int_add:Nn \l_@@_initial_i_int { #3 }
2888         \int_add:Nn \l_@@_initial_j_int { #4 }
2889         \bool_set_true:N \l_@@_initial_open_bool
2890         \bool_set_true:N \l_@@_stop_loop_bool
2891     }
2892     {
2893         \cs_if_exist:cTF
2894         {
2895             pgf @ sh @ ns @ \@@_env:
2896             - \int_use:N \l_@@_initial_i_int
2897             - \int_use:N \l_@@_initial_j_int
2898         }
2899         { \bool_set_true:N \l_@@_stop_loop_bool }
2900         {
2901             \cs_set:cpn
2902             {
2903                 @@ _ dotted _
2904                 \int_use:N \l_@@_initial_i_int -
2905                 \int_use:N \l_@@_initial_j_int
2906             }
2907             { }
2908         }
2909     }
2910 }
2911 }

```

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.

```

2912 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
2913 {
2914     { \int_use:N \l_@@_initial_i_int }
2915     { \int_use:N \l_@@_initial_j_int }
2916     { \int_use:N \l_@@_final_i_int }

```

```

2917         { \int_use:N \l_@@_final_j_int }
2918     }
2919 }

```

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).

```

2920 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
2921 {
2922     \int_set:Nn \l_@@_row_min_int 1
2923     \int_set:Nn \l_@@_col_min_int 1
2924     \int_set_eq:NN \l_@@_row_max_int \c@iRow
2925     \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`.

```

2926     \seq_map_inline:Nn \g_@@_submatrix_seq
2927     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
2928 }

```

#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.

```

2929 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
2930 {
2931     \bool_if:nT
2932     {
2933         \int_compare_p:n { #3 <= #1 }
2934         && \int_compare_p:n { #1 <= #5 }
2935         && \int_compare_p:n { #4 <= #2 }
2936         && \int_compare_p:n { #2 <= #6 }
2937     }
2938     {
2939         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
2940         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
2941         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
2942         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
2943     }
2944 }

```

```

2945 \cs_new_protected:Npn \@@_set_initial_coords:
2946 {
2947     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2948     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2949 }
2950 \cs_new_protected:Npn \@@_set_final_coords:
2951 {
2952     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2953     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2954 }
2955 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
2956 {
2957     \pgfpointanchor
2958     {
2959         \@@_env:
2960         - \int_use:N \l_@@_initial_i_int
2961         - \int_use:N \l_@@_initial_j_int
2962     }
2963     { #1 }
2964     \@@_set_initial_coords:
2965 }
2966 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1

```

```

2967 {
2968   \pgfpointanchor
2969   {
2970     \@@_env:
2971     - \int_use:N \l_@@_final_i_int
2972     - \int_use:N \l_@@_final_j_int
2973   }
2974   { #1 }
2975   \@@_set_final_coords:
2976 }
2977 \cs_new_protected:Npn \@@_open_x_initial_dim:
2978 {
2979   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
2980   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2981   {
2982     \cs_if_exist:cT
2983     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
2984     {
2985       \pgfpointanchor
2986       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
2987       { west }
2988       \dim_set:Nn \l_@@_x_initial_dim
2989       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
2990     }
2991   }

```

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

```

2992   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
2993   {
2994     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2995     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2996     \dim_add:Nn \l_@@_x_initial_dim \col@sep
2997   }
2998 }
2999 \cs_new_protected:Npn \@@_open_x_final_dim:
3000 {
3001   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3002   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3003   {
3004     \cs_if_exist:cT
3005     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3006     {
3007       \pgfpointanchor
3008       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3009       { east }
3010       \dim_set:Nn \l_@@_x_final_dim
3011       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3012     }
3013   }

```

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

```

3014   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3015   {
3016     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3017     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3018     \dim_sub:Nn \l_@@_x_final_dim \col@sep
3019   }
3020 }

```

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.

```

3021 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3

```



```

3022 {
3023   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3024   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3025   {
3026     \@@_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.

```

3027     \group_begin:
3028     \int_compare:nNnTF { #1 } = 0
3029     { \color { nicematrix-first-row } }
3030     {

```

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.

```

3031         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3032         { \color { nicematrix-last-row } }
3033     }
3034     \keys_set:nn { NiceMatrix / xdots } { #3 }
3035     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3036     \@@_actually_draw_Ldots:
3037   \group_end:
3038 }
3039 }

```

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`.

```

3040 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3041 {
3042   \bool_if:NTF \l_@@_initial_open_bool
3043   {
3044     \@@_open_x_initial_dim:
3045     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3046     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3047   }
3048   { \@@_set_initial_coords_from_anchor:n { base-east } }
3049   \bool_if:NTF \l_@@_final_open_bool
3050   {
3051     \@@_open_x_final_dim:
3052     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3053     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3054   }
3055   { \@@_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 (?).

```

3056   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3057   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3058   \@@_draw_line:
3059 }

```

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.

```

3060 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3061 {
3062   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3063   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3064   {
3065     \@@_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.

```

3066     \group_begin:
3067     \int_compare:nNnTF { #1 } = 0
3068     { \color { nicematrix-first-row } }
3069     {

```

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.

```

3070         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3071         { \color { nicematrix-last-row } }
3072     }
3073     \keys_set:nn { NiceMatrix / xdots } { #3 }
3074     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3075     \@@_actually_draw_Cdots:
3076     \group_end:
3077   }
3078 }

```

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`.

```

3079 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3080 {
3081   \bool_if:NTF \l_@@_initial_open_bool
3082   { \@@_open_x_initial_dim: }
3083   { \@@_set_initial_coords_from_anchor:n { mid-east } }
3084   \bool_if:NTF \l_@@_final_open_bool
3085   { \@@_open_x_final_dim: }
3086   { \@@_set_final_coords_from_anchor:n { mid-west } }
3087   \bool_lazy_and:nnTF
3088   \l_@@_initial_open_bool
3089   \l_@@_final_open_bool
3090   {
3091     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3092     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3093     \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
3094     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3095     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3096   }
3097   {
3098     \bool_if:NT \l_@@_initial_open_bool
3099     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3100     \bool_if:NT \l_@@_final_open_bool
3101     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3102   }

```

```

3103 \@@_draw_line:
3104 }
3105 \cs_new_protected:Npn \@@_open_y_initial_dim:
3106 {
3107   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3108   \dim_set:Nn \l_@@_y_initial_dim
3109     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3110   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3111     {
3112       \cs_if_exist:cT
3113         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3114         {
3115           \pgfpointanchor
3116             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3117             { north }
3118           \dim_set:Nn \l_@@_y_initial_dim
3119             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3120         }
3121     }
3122 }
3123 \cs_new_protected:Npn \@@_open_y_final_dim:
3124 {
3125   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3126   \dim_set:Nn \l_@@_y_final_dim
3127     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3128   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3129     {
3130       \cs_if_exist:cT
3131         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3132         {
3133           \pgfpointanchor
3134             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3135             { south }
3136           \dim_set:Nn \l_@@_y_final_dim
3137             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3138         }
3139     }
3140 }

```

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.

```

3141 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3142 {
3143   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3144   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3145   {
3146     \@@_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.

```

3147   \group_begin:
3148     \int_compare:nNnTF { #2 } = 0
3149     { \color { nicematrix-first-col } }
3150     {
3151       \int_compare:nNnT { #2 } = \l_@@_last_col_int
3152       { \color { nicematrix-last-col } }
3153     }
3154     \keys_set:nn { NiceMatrix / xdots } { #3 }
3155     \tl_if_empty:VF \l_@@_xdots_color_tl
3156     { \color { \l_@@_xdots_color_tl } }
3157     \@@_actually_draw_Vdots:
3158   \group_end:
3159 }
3160 }

```

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`.

```
3161 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3162 {
```

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

```
3163   \bool_set_false:N \l_tmpa_bool
```

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

```
3164   \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3165   {
3166     \@@_set_initial_coords_from_anchor:n { south-west }
3167     \@@_set_final_coords_from_anchor:n { north-west }
3168     \bool_set:Nn \l_tmpa_bool
3169     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3170   }
```

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

```
3171   \bool_if:NTF \l_@@_initial_open_bool
3172     \@@_open_y_initial_dim:
3173     { \@@_set_initial_coords_from_anchor:n { south } }
3174   \bool_if:NTF \l_@@_final_open_bool
3175     \@@_open_y_final_dim:
3176     { \@@_set_final_coords_from_anchor:n { north } }
3177   \bool_if:NTF \l_@@_initial_open_bool
3178   {
3179     \bool_if:NTF \l_@@_final_open_bool
3180     {
3181       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3182       \dim_set_eq:NN \l_tmpa_dim \pgf@x
3183       \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3184       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3185       \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.

```
3186       \int_compare:nNnT \l_@@_last_col_int > { -2 }
3187       {
3188         \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3189         {
3190           \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3191           \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3192           \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3193           \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3194         }
3195       }
3196     }
3197     { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3198   }
3199   {
3200     \bool_if:NTF \l_@@_final_open_bool
3201     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3202     {
```

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.

```

3203         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3204         {
3205             \dim_set:Nn \l_@@_x_initial_dim
3206             {
3207                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3208                 \l_@@_x_initial_dim \l_@@_x_final_dim
3209             }
3210             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3211         }
3212     }
3213 }
3214 \@@_draw_line:
3215 }
```

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.

```

3216 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3217 {
3218     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3219     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3220     {
3221         \@@_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.

```

3222         \group_begin:
3223         \keys_set:nn { NiceMatrix / xdots } { #3 }
3224         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3225         \@@_actually_draw_Ddots:
3226         \group_end:
3227     }
3228 }
```

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.`

```

3229 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3230 {
3231     \bool_if:NTF \l_@@_initial_open_bool
3232     {
3233         \@@_open_y_initial_dim:
3234         % \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3235         % \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3236         \@@_open_x_initial_dim:
3237     }
3238     { \@@_set_initial_coords_from_anchor:n { south-east } }
3239     \bool_if:NTF \l_@@_final_open_bool
```

```

3240 {
3241   % \@@_open_y_final_dim:
3242   % \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3243   \@@_open_x_final_dim:
3244   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3245 }
3246 { \@@_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.

```

3247   \bool_if:NT \l_@@_parallelize_diags_bool
3248   {
3249     \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).

```

3250     \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  $\Delta_x$  and the  $\Delta_y$  of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

3251     {
3252       \dim_gset:Nn \g_@@_delta_x_one_dim
3253       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3254       \dim_gset:Nn \g_@@_delta_y_one_dim
3255       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3256     }

```

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`.

```

3257     {
3258       \dim_set:Nn \l_@@_y_final_dim
3259       {
3260         \l_@@_y_initial_dim +
3261         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3262         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3263       }
3264     }
3265   }
3266   \@@_draw_line:
3267 }

```

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.

```

3268 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3269 {
3270   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3271   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3272   {
3273     \@@_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.

```

3274     \group_begin:
3275     \keys_set:nn { NiceMatrix / xdots } { #3 }
3276     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3277     \@@_actually_draw_Iddots:
3278     \group_end:
3279   }
3280 }

```

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.

```

3281 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3282 {
3283   \bool_if:NTF \l_@@_initial_open_bool
3284   {
3285     \@@_open_y_initial_dim:
3286     \@@_open_x_initial_dim:
3287   }
3288   { \@@_set_initial_coords_from_anchor:n { south-west } }
3289   \bool_if:NTF \l_@@_final_open_bool
3290   {
3291     \@@_open_y_final_dim:
3292     \@@_open_x_final_dim:
3293   }
3294   { \@@_set_final_coords_from_anchor:n { north-east } }
3295   \bool_if:NT \l_@@_parallelize_diags_bool
3296   {
3297     \int_gincr:N \g_@@_iddots_int
3298     \int_compare:nNnTF \g_@@_iddots_int = 1
3299     {
3300       \dim_gset:Nn \g_@@_delta_x_two_dim
3301       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3302       \dim_gset:Nn \g_@@_delta_y_two_dim
3303       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3304     }
3305     {
3306       \dim_set:Nn \l_@@_y_final_dim
3307       {
3308         \l_@@_y_initial_dim +
3309         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3310         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3311       }
3312     }
3313   }
3314   \@@_draw_line:
3315 }

```

## The actual instructions for drawing the dotted line 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

```

3316 \cs_new_protected:Npn \@@_draw_line:
3317 {
3318   \pgfrememberpicturepositiononpagetrue
3319   \pgf@relevantforpicturesizefalse
3320   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
3321     \@@_draw_standard_dotted_line:
3322     \@@_draw_non_standard_dotted_line:
3323 }

```

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.

```

3324 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
3325 {
3326   \begin { scope }
3327   \exp_args:No \@@_draw_non_standard_dotted_line:n
3328     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3329 }

```

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_non_standard_dotted_line:n` is, in fact, the list of options.

```

3330 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
3331 {
3332   \@@_draw_non_standard_dotted_line:nVV
3333     { #1 }
3334     \l_@@_xdots_up_tl
3335     \l_@@_xdots_down_tl
3336 }
3337 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:nnn #1 #2 #3
3338 {
3339   \draw
3340     [
3341       #1 ,
3342       shorten-> = \l_@@_xdots_shorten_dim ,
3343       shorten-< = \l_@@_xdots_shorten_dim ,
3344     ]
3345     ( \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).

```

3346   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3347   node [ sloped , below ] { $ \scriptstyle #3 $ }
3348   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3349 \end { scope }
3350 }
3351 \cs_generate_variant:Nn \@@_draw_non_standard_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).

```

3352 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3353 {
3354   \bool_lazy_and:nnF
3355     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3356     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3357   {
3358     \pgfscope
3359     \pgftransformshift
3360       {
3361         \pgfpointlineattime { 0.5 }
3362         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3363         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3364       }

```



```

3365 \pgftransformrotate
3366 {
3367     \fp_eval:n
3368     {
3369         atand
3370         (
3371             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3372             \l_@@_x_final_dim - \l_@@_x_initial_dim
3373         )
3374     }
3375 }
3376 \pgfnode
3377 { rectangle }
3378 { south }
3379 {
3380     \c_math_toggle_token
3381     \scriptstyle \l_@@_xdots_up_tl
3382     \c_math_toggle_token
3383 }
3384 { }
3385 { \pgfusepath { } }
3386 \pgfnode
3387 { rectangle }
3388 { north }
3389 {
3390     \c_math_toggle_token
3391     \scriptstyle \l_@@_xdots_down_tl
3392     \c_math_toggle_token
3393 }
3394 { }
3395 { \pgfusepath { } }
3396 \endpgfscope
3397 }
3398 \group_begin:

```

The dimension `\l_@@_l_dim` is the length  $\ell$  of the line to draw. We use the floating point reals of `expl3` to compute this length.

```

3399 \dim_zero_new:N \l_@@_l_dim
3400 \dim_set:Nn \l_@@_l_dim
3401 {
3402     \fp_to_dim:n
3403     {
3404         sqrt
3405         (
3406             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3407             +
3408             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3409         )
3410     }
3411 }

```

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.

```

3412 \bool_lazy_or:nnF
3413 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3414 { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3415 \@@_draw_standard_dotted_line_i:
3416 \group_end:
3417 }
3418 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3419 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3420 {

```

The number of dots will be  $\backslash l\_tmpa\_int + 1$ .

```

3421 \bool_if:NTF \l_@@_initial_open_bool
3422 {
3423   \bool_if:NTF \l_@@_final_open_bool
3424   {
3425     \int_set:Nn \l_tmpa_int
3426     { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3427   }
3428   {
3429     \int_set:Nn \l_tmpa_int
3430     {
3431       \dim_ratio:nn
3432       { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3433       \l_@@_inter_dots_dim
3434     }
3435   }
3436 }
3437 {
3438   \bool_if:NTF \l_@@_final_open_bool
3439   {
3440     \int_set:Nn \l_tmpa_int
3441     {
3442       \dim_ratio:nn
3443       { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3444       \l_@@_inter_dots_dim
3445     }
3446   }
3447   {
3448     \int_set:Nn \l_tmpa_int
3449     {
3450       \dim_ratio:nn
3451       { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3452       \l_@@_inter_dots_dim
3453     }
3454   }
3455 }

```

The dimensions  $\backslash l\_tmpa\_dim$  and  $\backslash l\_tmpb\_dim$  are the coordinates of the vector between two dots in the dotted line.

```

3456 \dim_set:Nn \l_tmpa_dim
3457 {
3458   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3459   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3460 }
3461 \dim_set:Nn \l_tmpb_dim
3462 {
3463   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3464   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3465 }

```

The length  $\ell$  is the length of the dotted line. We note  $\Delta$  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  $\backslash l\_tmpb\_int$ .

```

3466 \int_set:Nn \l_tmpb_int
3467 {
3468   \bool_if:NTF \l_@@_initial_open_bool
3469   { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3470   { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3471 }

```

In the loop over the dots, the dimensions  $\backslash l\_@@\_x\_initial\_dim$  and  $\backslash l\_@@\_y\_initial\_dim$  will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

3472 \dim_gadd:Nn \l_@@_x_initial_dim
3473 {
3474   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3475   \dim_ratio:nn
3476   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3477   { 2 \l_@@_l_dim }
3478   * \l_tmpb_int
3479 }
3480 \dim_gadd:Nn \l_@@_y_initial_dim
3481 {
3482   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3483   \dim_ratio:nn
3484   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3485   { 2 \l_@@_l_dim }
3486   * \l_tmpb_int
3487 }
3488 \pgf@relevantforpicturesizefalse
3489 \int_step_inline:nnn 0 \l_tmpa_int
3490 {
3491   \pgfpathcircle
3492   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3493   { \l_@@_radius_dim }
3494   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3495   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3496 }
3497 \pgfusepathqfill
3498 }

```

## 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.

```

3499 \AtBeginDocument
3500 {
3501   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3502   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3503   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3504   {
3505     \int_compare:nNnTF \c@jCol = 0
3506     { \@@_error:nn { in~first~col } \Ldots }
3507     {
3508       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3509       { \@@_error:nn { in~last~col } \Ldots }
3510       {
3511         \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
3512         { #1 , down = #2 , up = #3 }
3513       }
3514     }
3515     \bool_if:NF \l_@@_nullify_dots_bool
3516     { \phantom { \ensuremath { \@@_old_ldots } } }
3517     \bool_gset_true:N \g_@@_empty_cell_bool
3518   }

```

```

3519 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3520 {
3521   \int_compare:nNnTF \c@jCol = 0
3522   { \@@_error:nn { in~first~col } \Cdots }
3523   {
3524     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3525     { \@@_error:nn { in~last~col } \Cdots }
3526     {
3527       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
3528       { #1 , down = #2 , up = #3 }
3529     }
3530   }
3531   \bool_if:NF \l_@@_nullify_dots_bool
3532   { \phantom { \ensuremath { \@@_old_cdots } } }
3533   \bool_gset_true:N \g_@@_empty_cell_bool
3534 }

3535 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
3536 {
3537   \int_compare:nNnTF \c@iRow = 0
3538   { \@@_error:nn { in~first~row } \Vdots }
3539   {
3540     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
3541     { \@@_error:nn { in~last~row } \Vdots }
3542     {
3543       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
3544       { #1 , down = #2 , up = #3 }
3545     }
3546   }
3547   \bool_if:NF \l_@@_nullify_dots_bool
3548   { \phantom { \ensuremath { \@@_old_vdots } } }
3549   \bool_gset_true:N \g_@@_empty_cell_bool
3550 }

3551 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
3552 {
3553   \int_case:nnF \c@iRow
3554   {
3555     0 { \@@_error:nn { in~first~row } \Ddots }
3556     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
3557   }
3558   {
3559     \int_case:nnF \c@jCol
3560     {
3561       0 { \@@_error:nn { in~first~col } \Ddots }
3562       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
3563     }
3564     {
3565       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3566       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
3567       { #1 , down = #2 , up = #3 }
3568     }
3569   }
3570 }
3571 \bool_if:NF \l_@@_nullify_dots_bool
3572 { \phantom { \ensuremath { \@@_old_ddots } } }
3573 \bool_gset_true:N \g_@@_empty_cell_bool
3574 }

3575 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
3576 {

```

```

3577 \int_case:nnF \c@iRow
3578 {
3579     0 { \@@_error:nn { in~first~row } \Iddots }
3580     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
3581 }
3582 {
3583     \int_case:nnF \c@jCol
3584     {
3585         0 { \@@_error:nn { in~first~col } \Iddots }
3586         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
3587     }
3588     {
3589         \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3590         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
3591         { #1 , down = #2 , up = #3 }
3592     }
3593 }
3594 \bool_if:NF \l_@@_nullify_dots_bool
3595 { \phantom { \ensuremath { \@@_old_iddots } } }
3596 \bool_gset_true:N \g_@@_empty_cell_bool
3597 }
3598 }

```

End of the `\AtBeginDocument`.

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

```

3599 \keys_define:nn { NiceMatrix / Ddots }
3600 {
3601     draw-first .bool_set:N = \l_@@_draw_first_bool ,
3602     draw-first .default:n = true ,
3603     draw-first .value_forbidden:n = true
3604 }

```

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

```

3605 \cs_new_protected:Npn \@@_Hspace:
3606 {
3607     \bool_gset_true:N \g_@@_empty_cell_bool
3608     \hspace
3609 }

```

In the environment `{NiceArray}`, the command `\multicolumn` will be linked to the following command `\@@_multicolumn:nnn`.

```

3610 \cs_set_eq:NN \@@_old_multicolumn \multicolumn
3611 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3612 {

```

We have to act in an expandable way since it will begin by a `\multicolumn`.

```

3613 \exp_args:NNe
3614 \@@_old_multicolumn
3615 { #1 }
3616 {
3617     \exp_args:Ne \str_case:nn { \str_foldcase:n { #2 } }
3618     {
3619         l { > \@@_Cell: l < \@@_end_Cell: }
3620         r { > \@@_Cell: r < \@@_end_Cell: }
3621         c { > \@@_Cell: c < \@@_end_Cell: }
3622         { l | } { > \@@_Cell: l < \@@_end_Cell: | }
3623         { r | } { > \@@_Cell: r < \@@_end_Cell: | }
3624         { c | } { > \@@_Cell: c < \@@_end_Cell: | }
3625         { | l } { | > \@@_Cell: l < \@@_end_Cell: }
3626         { | r } { | > \@@_Cell: r < \@@_end_Cell: }
3627         { | c } { | > \@@_Cell: c < \@@_end_Cell: }
3628         { | l | } { | > \@@_Cell: l < \@@_end_Cell: | }

```

```

3629         { | r | } { | > \@@_Cell: r < \@@_end_Cell: | }
3630         { | c | } { | > \@@_Cell: c < \@@_end_Cell: | }
3631     }
3632 }
3633 { #3 }

```

The `\peek_remove_spaces:n` is mandatory.

```

3634 \peek_remove_spaces:n
3635 {
3636     \int_compare:nNnT #1 > 1
3637     {
3638         \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
3639         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3640         \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
3641         \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
3642         {
3643             { \int_use:N \c@iRow }
3644             { \int_use:N \c@jCol }
3645             { \int_use:N \c@iRow }
3646             { \int_eval:n { \c@jCol + #1 - 1 } }
3647         }
3648     }
3649     \int_gadd:Nn \c@jCol { #1 - 1 }
3650     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3651     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3652 }
3653 }

```

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`.

```

3654 \cs_new:Npn \@@_Hdotsfor:
3655 {
3656     \bool_lazy_and:nnTF
3657     { \int_compare_p:nNn \c@jCol = 0 }
3658     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
3659     {
3660         \bool_if:NTF \g_@@_after_col_zero_bool
3661         {
3662             \multicolumn { 1 } { c } { }
3663             \@@_Hdotsfor_i
3664         }
3665         { \@@_fatal:n { Hdotsfor~in~col~0 } }
3666     }
3667     {
3668         \multicolumn { 1 } { c } { }
3669         \@@_Hdotsfor_i
3670     }
3671 }

```

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:`).

```

3672 \AtBeginDocument
3673 {
3674     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3675     \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.

```

3676     \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
3677     {
3678         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl

```

```

3679     {
3680         \@@_Hdotsfor:nnnn
3681         { \int_use:N \c@iRow }
3682         { \int_use:N \c@jCol }
3683         { #2 }
3684         {
3685             #1 , #3 ,
3686             down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3687         }
3688     }
3689     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
3690 }
3691 }

```

Enf of \AtBeginDocument.

```

3692 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
3693 {
3694     \bool_set_false:N \l_@@_initial_open_bool
3695     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

3696     \int_set:Nn \l_@@_initial_i_int { #1 }
3697     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

3698     \int_compare:nNnTF { #2 } = 1
3699     {
3700         \int_set:Nn \l_@@_initial_j_int 1
3701         \bool_set_true:N \l_@@_initial_open_bool
3702     }
3703     {
3704         \cs_if_exist:cTF
3705         {
3706             pgf @ sh @ ns @ \@@_env:
3707             - \int_use:N \l_@@_initial_i_int
3708             - \int_eval:n { #2 - 1 }
3709         }
3710         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
3711         {
3712             \int_set:Nn \l_@@_initial_j_int { #2 }
3713             \bool_set_true:N \l_@@_initial_open_bool
3714         }
3715     }
3716     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
3717     {
3718         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3719         \bool_set_true:N \l_@@_final_open_bool
3720     }
3721     {
3722         \cs_if_exist:cTF
3723         {
3724             pgf @ sh @ ns @ \@@_env:
3725             - \int_use:N \l_@@_final_i_int
3726             - \int_eval:n { #2 + #3 }
3727         }
3728         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
3729         {
3730             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3731             \bool_set_true:N \l_@@_final_open_bool
3732         }
3733     }
3734     \group_begin:
3735     \int_compare:nNnTF { #1 } = 0

```

```

3736     { \color { nicematrix-first-row } }
3737     {
3738         \int_compare:nNnT { #1 } = \g_@@_row_total_int
3739         { \color { nicematrix-last-row } }
3740     }
3741     \keys_set:nn { NiceMatrix / xdots } { #4 }
3742     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3743     \@@_actually_draw_ldots:
3744     \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).

```

3745     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
3746     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
3747 }

```

```

3748 \AtBeginDocument
3749 {
3750     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3751     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3752     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
3753     {
3754         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
3755         {
3756             \@@_Vdotsfor:nnnn
3757             { \int_use:N \c@iRow }
3758             { \int_use:N \c@jCol }
3759             { #2 }
3760             {
3761                 #1 , #3 ,
3762                 down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3763             }
3764         }
3765     }
3766 }

```

Enf of `\AtBeginDocument`.

```

3767 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
3768 {
3769     \bool_set_false:N \l_@@_initial_open_bool
3770     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

3771     \int_set:Nn \l_@@_initial_j_int { #2 }
3772     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

3773     \int_compare:nNnTF #1 = 1
3774     {
3775         \int_set:Nn \l_@@_initial_i_int 1
3776         \bool_set_true:N \l_@@_initial_open_bool
3777     }
3778     {
3779         \cs_if_exist:cTF
3780         {
3781             pgf @ sh @ ns @ \@@_env:
3782             - \int_eval:n { #1 - 1 }
3783             - \int_use:N \l_@@_initial_j_int
3784         }
3785         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
3786         {
3787             \int_set:Nn \l_@@_initial_i_int { #1 }

```



```

3788         \bool_set_true:N \l_@@_initial_open_bool
3789     }
3790 }
3791 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
3792 {
3793     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3794     \bool_set_true:N \l_@@_final_open_bool
3795 }
3796 {
3797     \cs_if_exist:cTF
3798     {
3799         pgf @ sh @ ns @ \@@_env:
3800         - \int_eval:n { #1 + #3 }
3801         - \int_use:N \l_@@_final_j_int
3802     }
3803     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
3804     {
3805         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3806         \bool_set_true:N \l_@@_final_open_bool
3807     }
3808 }
3809 \group_begin:
3810 \int_compare:nNnTF { #2 } = 0
3811 { \color { nicematrix-first-col } }
3812 {
3813     \int_compare:nNnT { #2 } = \g_@@_col_total_int
3814     { \color { nicematrix-last-col } }
3815 }
3816 \keys_set:nn { NiceMatrix / xdots } { #4 }
3817 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3818 \@@_actually_draw_Vdots:
3819 \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).

```

3820     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
3821     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
3822 }

```

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

```

3823 \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).<sup>60</sup>

```

3824 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
3825 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

---

<sup>60</sup>Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

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

```

3826 \AtBeginDocument
3827 {
3828   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
3829   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3830   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
3831     {
3832       \group_begin:
3833       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
3834       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3835       \use:e
3836       {
3837         \@@_line_i:nn
3838         { \@@_double_int_eval:n #2 \q_stop }
3839         { \@@_double_int_eval:n #3 \q_stop }
3840       }
3841       \group_end:
3842     }
3843 }
3844 \cs_new_protected:Npn \@@_line_i:nn #1 #2
3845 {
3846   \bool_set_false:N \l_@@_initial_open_bool
3847   \bool_set_false:N \l_@@_final_open_bool
3848   \bool_if:nTF
3849     {
3850       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
3851       ||
3852       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
3853     }
3854     {
3855       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
3856     }
3857     { \@@_draw_line_ii:nn { #1 } { #2 } }
3858 }
3859 \AtBeginDocument
3860 {
3861   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
3862   {

```

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:`.

```

3863   \c_@@_pgfortikzpicture_tl
3864   \@@_draw_line_iii:nn { #1 } { #2 }
3865   \c_@@_endpgfortikzpicture_tl
3866 }
3867 }

```

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

```

3868 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
3869 {
3870   \pgfrememberpicturepositiononpagetrue
3871   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
3872   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3873   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3874   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
3875   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3876   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3877   \@@_draw_line:
3878 }

```

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).

## 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` and `\@@_rectanglecolor` (which are linked to `\rowcolor`, `\columncolor` and `\rectanglecolor` before the execution of the `code-before`) 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 `\@@_rowcolor:n`, `\@@_columncolor:n` and `\@@_rectanglecolor:nn` (corresponding of `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor`).

`\bigskip #1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_color_seq` 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).

```
3879 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
3880 {
```

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.

```
3881 \int_zero:N \l_tmpa_int
3882 \seq_map_indexed_inline:Nn \g_@@_colors_seq
3883 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
3884 \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

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

```
3885 {
3886 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
3887 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
3888 }
```

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

```
3889 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
3890 }
```

```
3891 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
```

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`).

```
3892 \cs_new_protected:Npn \@@_actually_color:
3893 {
3894 \pgfpicture
3895 \pgf@relevantforpicturesizefalse
3896 \seq_map_indexed_inline:Nn \g_@@_colors_seq
3897 {
3898 \color ##2
3899 \use:c { g_@@_color _ ##1 _tl }
3900 \tl_gclear:c { g_@@_color _ ##1 _tl }
3901 \pgfusepath { fill }
```

```

3902     }
3903     \endpgfpicture
3904 }

3905 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
3906 {
3907     \tl_set:Nn \l_tmpa_tl { #1 }
3908     \tl_set:Nn \l_tmpb_tl { #2 }
3909 }

```

Here is an example : \@@\_rowcolor {red!15} {1,3,5-7,10-}

```

3910 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
3911 {
3912     \tl_if_blank:nF { #2 }
3913     {
3914         \@@_add_to_colors_seq:xn
3915         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3916         { \@@_rowcolor:n { #3 } }
3917     }
3918 }

3919 \cs_new_protected:Npn \@@_rowcolor:n #1
3920 {
3921     \tl_set:Nn \l_@@_rows_tl { #1 }
3922     \tl_set:Nn \l_@@_cols_tl { - }

```

The command \@@\_cartesian\_path: takes in two implicit arguments: \l\_@@\_cols\_tl and \l\_@@\_rows\_tl.

```

3923     \@@_cartesian_path:
3924 }

```

Here an example : \@@\_columncolor:nn {red!15} {1,3,5-7,10-}

```

3925 \NewDocumentCommand \@@_columncolor { 0 { } m m }
3926 {
3927     \tl_if_blank:nF { #2 }
3928     {
3929         \@@_add_to_colors_seq:xn
3930         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3931         { \@@_columncolor:n { #3 } }
3932     }
3933 }

3934 \cs_new_protected:Npn \@@_columncolor:n #1
3935 {
3936     \tl_set:Nn \l_@@_rows_tl { - }
3937     \tl_set:Nn \l_@@_cols_tl { #1 }

```

The command \@@\_cartesian\_path: takes in two implicit arguments: \l\_@@\_cols\_tl and \l\_@@\_rows\_tl.

```

3938     \@@_cartesian_path:
3939 }

```

Here is an example : \@@\_rectanglecolor{red!15}{2-3}{5-6}

```

3940 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
3941 {
3942     \tl_if_blank:nF { #2 }
3943     {
3944         \@@_add_to_colors_seq:xn
3945         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3946         { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
3947     }
3948 }

```

The last argument is the radius of the corners of the rectangle.

```

3949 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
3950 {
3951   \tl_if_blank:nF { #2 }
3952   {
3953     \@@_add_to_colors_seq:xn
3954     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3955     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
3956   }
3957 }

```

The last argument is the radius of the corners of the rectangle.

```

3958 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
3959 {
3960   \@@_cut_on_hyphen:w #1 \q_stop
3961   \tl_clear_new:N \l_tmpc_tl
3962   \tl_clear_new:N \l_tmpd_tl
3963   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
3964   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
3965   \@@_cut_on_hyphen:w #2 \q_stop
3966   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
3967   \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`.

```

3968   \@@_cartesian_path:n { #3 }
3969 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

3970 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
3971 {
3972   \clist_map_inline:nn { #3 }
3973   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
3974 }

```

```

3975 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
3976 {
3977   \int_step_inline:nn { \int_use:N \c@iRow }
3978   {
3979     \int_step_inline:nn { \int_use:N \c@jCol }
3980     {
3981       \int_if_even:nTF { ####1 + ##1 }
3982       { \@@_cellcolor [ #1 ] { #2 } }
3983       { \@@_cellcolor [ #1 ] { #3 } }
3984     } { ##1 - ####1 }
3985   }
3986 }
3987 }

```

```

3988 \keys_define:nn { NiceMatrix / arraycolor }
3989 { 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.

```

3990 \NewDocumentCommand \@@_arraycolor { 0 { } m 0 { } }
3991 {
3992   \keys_set:nn { NiceMatrix / arraycolor } { #3 }
3993   \@@_rectanglecolor [ #1 ] { #2 }
3994   { 1 - 1 }
3995   { \int_use:N \c@iRow - \int_use:N \c@jCol }
3996 }

```

```

3997 \keys_define:nn { NiceMatrix / rowcolors }
3998 {
3999   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4000   respect-blocks .default:n = true ,
4001   cols .tl_set:N = \l_@@_cols_tl ,
4002   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4003   restart .default:n = true ,
4004   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4005 }

```

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 first color ; #4 is the second color ; #5 is for the optional list of pairs key-value.

```

4006 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4007 {

```

The group is for the options.

```

4008   \group_begin:
4009   \tl_clear_new:N \l_@@_cols_tl
4010   \tl_set:Nn \l_@@_cols_tl { - }
4011   \keys_set:nn { NiceMatrix / rowcolors } { #5 }

```

The boolean `\l_tmpa_bool` will indicate whereas we are in a row of the first color or of the second color.

```

4012   \bool_set_true:N \l_tmpa_bool
4013   \bool_lazy_and:nnT
4014     \l_@@_respect_blocks_bool
4015     {
4016       \cs_if_exist_p:c
4017         { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } }
4018   {

```

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`).

```

4019     \seq_set_eq:Nc \l_tmpb_seq
4020     { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
4021     \seq_set_filter:Nn \l_tmpa_seq \l_tmpb_seq
4022     { \@@_not_in_exterior_p:nnnn ##1 }
4023   }
4024   \pgfpicture
4025   \pgf@relevantforpicturesizefalse
4026   \clist_map_inline:nn { #2 }
4027   {
4028     \tl_set:Nn \l_tmpa_tl { ##1 }
4029     \tl_if_in:NnTF \l_tmpa_tl { - }
4030       { \@@_cut_on_hyphen:w ##1 \q_stop }
4031       { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

The counter `\l_tmpa_int` will be the index of the loop.

```

4032   \int_set:Nn \l_tmpa_int \l_tmpa_tl
4033   \bool_if:NnTF \l_@@_rowcolors_restart_bool
4034     { \bool_set_true:N \l_tmpa_bool }
4035     { \bool_set:Nn \l_tmpa_bool { \int_if_odd_p:n { \l_tmpa_tl } } }
4036   \int_zero_new:N \l_tmppc_int
4037   \int_set:Nn \l_tmppc_int \l_tmppb_tl
4038   \int_do_until:nNnn \l_tmpa_int > \l_tmppc_int
4039   {

```

We will compute in `\l_tmppb_int` the last row of the “block”.

```

4040     \int_set_eq:NN \l_tmppb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

4041         \bool_lazy_and:nnT
4042         \l_@@_respect_blocks_bool
4043         {
4044             \cs_if_exist_p:c
4045             { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
4046         }
4047         {
4048             \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
4049             { \@@_intersect_our_row_p:nnnn ###1 }
4050             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4051     }
4052     \tl_set:Nx \l_@@_rows_tl
4053     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
4054     \bool_if:NTF \l_tmpa_bool
4055     {
4056         \tl_if_blank:nF { #3 }
4057         {
4058             \tl_if_empty:nTF { #1 }
4059             \color
4060             { \color [ #1 ] }
4061             { #3 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4062         \@@_cartesian_path:
4063         \pgfusepath { fill }
4064     }
4065     \bool_set_false:N \l_tmpa_bool
4066 }
4067 {
4068     \tl_if_blank:nF { #4 }
4069     {
4070         \tl_if_empty:nTF { #1 }
4071         \color
4072         { \color [ #1 ] }
4073         { #4 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

4074         \@@_cartesian_path:
4075         \pgfusepath { fill }
4076     }
4077     \bool_set_true:N \l_tmpa_bool
4078 }
4079     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4080 }
4081 }
4082 \endpgfpicture
4083 \group_end:
4084 }

```

```

4085 \cs_new_protected:Npn \@@_rowcolors_i:nnnn #1 #2 #3 #4
4086 {
4087     \int_compare:nNnT { #3 } > \l_tmpb_int
4088     { \int_set:Nn \l_tmpb_int { #3 } }
4089 }

```

```

4090 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnn p
4091 {
4092     \bool_lazy_or:nnTF

```

```

4093     { \int_compare_p:nNn { #4 } = \c_zero_int }
4094     { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
4095     \prg_return_false:
4096     \prg_return_true:
4097 }

```

The following command return true when the block intersects the row \l\_tmpa\_int.

```

4098 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnn p
4099 {
4100   \bool_if:nTF
4101   {
4102     \int_compare_p:n { #1 <= \l_tmpa_int }
4103     &&
4104     \int_compare_p:n { \l_tmpa_int <= #3 }
4105   }
4106   \prg_return_true:
4107   \prg_return_false:
4108 }

```

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).

```

4109 \cs_new_protected:Npn \@@_cartesian_path:n #1
4110 {
4111   \bool_lazy_and:nnT
4112   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4113   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4114   {
4115     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4116     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4117   }

```

We begin the loop over the columns.

```

4118   \clist_map_inline:Nn \l_@@_cols_tl
4119   {
4120     \tl_set:Nn \l_tmpa_tl { ##1 }
4121     \tl_if_in:NnTF \l_tmpa_tl { - }
4122     { \@@_cut_on_hyphen:w ##1 \q_stop }
4123     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4124     \bool_lazy_or:nnT
4125     { \tl_if_blank_p:V \l_tmpa_tl }
4126     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4127     { \tl_set:Nn \l_tmpa_tl { 1 } }
4128     \bool_lazy_or:nnT
4129     { \tl_if_blank_p:V \l_tmpb_tl }
4130     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4131     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4132     \int_compare:nNnT \l_tmpb_tl > \c@jCol
4133     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

\l\_tmpc\_tl will contain the number of column.

```

4134     \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.

```

4135     \@@_qpoint:n { col - \l_tmpa_tl }
4136     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4137     { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4138     { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }

```



```

4139 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4140 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4141 \clist_map_inline:Nn \l_@@_rows_tl
4142 {
4143   \tl_set:Nn \l_tmpa_tl { ####1 }
4144   \tl_if_in:NnTF \l_tmpa_tl { - }
4145   { \@@_cut_on_hyphen:w ####1 \q_stop }
4146   { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
4147   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4148   \tl_if_empty:NT \l_tmpb_tl
4149   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4150   \int_compare:nNnT \l_tmpb_tl > \c@iRow
4151   { \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.

```

4152 \seq_if_in:NxF \l_@@_corners_cells_seq
4153 { \l_tmpa_tl - \l_tmpc_tl }
4154 {
4155   \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
4156   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4157   \@@_qpoint:n { row - \l_tmpa_tl }
4158   \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4159   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4160   \pgfpathrectanglecorners
4161   { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4162   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4163 }
4164 }
4165 }
4166 }

```

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).

```

4167 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

The following command will be used only with \l\_@@\_cols\_tl and \c@jCol (first case) or with \l\_@@\_rows\_tl and \c@iRow (second case). For instance, with \l\_@@\_cols\_tl equal to 2,4-6,8-\* and \c@jCol equal to 10, the clist \l\_@@\_cols\_tl will be replaced by 2,4,5,6,8,9,10.

```

4168 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4169 {
4170   \clist_set_eq:NN \l_tmpa_clist #1
4171   \clist_clear:N #1
4172   \clist_map_inline:Nn \l_tmpa_clist
4173   {
4174     \tl_set:Nn \l_tmpa_tl { ##1 }
4175     \tl_if_in:NnTF \l_tmpa_tl { - }
4176     { \@@_cut_on_hyphen:w ##1 \q_stop }
4177     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4178     \bool_lazy_or:nnT
4179     { \tl_if_blank_p:V \l_tmpa_tl }
4180     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4181     { \tl_set:Nn \l_tmpa_tl { 1 } }
4182     \bool_lazy_or:nnT
4183     { \tl_if_blank_p:V \l_tmpb_tl }
4184     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4185     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4186     \int_compare:nNnT \l_tmpb_tl > #2
4187     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4188     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4189     { \clist_put_right:Nn #1 { ####1 } }
4190   }
4191 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```

4192 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4193 {
4194   \peek_remove_spaces:n
4195   {
4196     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4197     {

```

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`).

```

4198       \cellcolor [ #1 ] { \exp_not:n { #2 } }
4199       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4200     }
4201   }
4202 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the tabular.

```

4203 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4204 {
4205   \peek_remove_spaces:n
4206   {
4207     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4208     {
4209       \exp_not:N \rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4210       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4211       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4212     }
4213   }
4214 }

```

```

4215 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4216 {

```

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).

```

4217   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4218   {

```

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).

```

4219     \tl_gput_left:Nx \g_nicematrix_code_before_tl
4220     {
4221       \exp_not:N \columncolor [ #1 ]
4222       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4223     }
4224   }
4225 }

```

## 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`.

```
4226 \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`.

```
4227 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4228 {
4229   \int_compare:nNnTF \l_@@_first_col_int = 0
4230   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4231   {
4232     \int_compare:nNnTF \c@jCol = 0
4233     {
4234       \int_compare:nNnF \c@iRow = { -1 }
4235       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4236     }
4237     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4238   }
4239 }
```

This definition may seem complicated by 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.

```
4240 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4241 {
4242   \int_compare:nNnF \c@iRow = 0
4243   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4244 }
```

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 `|`.

```
4245 \cs_new_protected:Npn \@@_vline:nn #1 #2
4246 {
```

The following test is for the case where the user don't use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```
4247   \int_compare:nNnT { #1 } < { \c@jCol + 2 }
4248   {
4249     \pgfpicture
4250     \@@_vline_i:nn { #1 } { #2 }
4251     \endpgfpicture
4252   }
4253 }
4254 \cs_new_protected:Npn \@@_vline_i:nn #1 #2
4255 {
```

`\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_tmpe_tl`.

```
4256   \tl_set:Nx \l_tmpb_tl { #1 }
4257   \tl_clear_new:N \l_tmpe_tl
4258   \int_step_variable:nNn \c@iRow \l_tmpa_tl
4259   {
```

The boolean `\g_tmpe_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_tmpe_bool` to `false` and the small vertical rule won't be drawn.

```
4260     \bool_gset_true:N \g_tmpe_bool
4261     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4262     { \@@_test_vline_in_block:nnnn ##1 }
```

```

4263 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4264 { \@@_test_vline_in_block:nnnn #1 }
4265 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4266 { \@@_test_vline_in_stroken_block:nnnn #1 }
4267 \clist_if_empty:NF \l_@@_corners_clist
4268 \@@_test_in_corner_v:
4269 \bool_if:NTF \g_tmpa_bool
4270 {
4271 \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4272 { \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl }
4273 }
4274 {
4275 \tl_if_empty:NF \l_tmpc_tl
4276 {
4277 \@@_vline_ii:nnnn
4278 { #1 }
4279 { #2 }
4280 \l_tmpc_tl
4281 { \int_eval:n { \l_tmpa_tl - 1 } }
4282 \tl_clear:N \l_tmpc_tl
4283 }
4284 }
4285 }
4286 \tl_if_empty:NF \l_tmpc_tl
4287 {
4288 \@@_vline_ii:nnnn
4289 { #1 }
4290 { #2 }
4291 \l_tmpc_tl
4292 { \int_use:N \c@iRow }
4293 \tl_clear:N \l_tmpc_tl
4294 }
4295 }

4296 \cs_new_protected:Npn \@@_test_in_corner_v:
4297 {
4298 \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
4299 {
4300 \seq_if_in:NxT
4301 \l_@@_corners_cells_seq
4302 { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4303 { \bool_set_false:N \g_tmpa_bool }
4304 }
4305 {
4306 \seq_if_in:NxT
4307 \l_@@_corners_cells_seq
4308 { \l_tmpa_tl - \l_tmpb_tl }
4309 {
4310 \int_compare:nNnTF \l_tmpb_tl = 1
4311 { \bool_set_false:N \g_tmpa_bool }
4312 {
4313 \seq_if_in:NxT
4314 \l_@@_corners_cells_seq
4315 { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4316 { \bool_set_false:N \g_tmpa_bool }
4317 }
4318 }
4319 }
4320 }

```

#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.

```

4321 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
4322 {
4323   \pgfrememberpicturepositiononpagetrue
4324   \pgf@relevantforpicturesizefalse
4325   \@@_qpoint:n { row - #3 }
4326   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4327   \@@_qpoint:n { col - #1 }
4328   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4329   \@@_qpoint:n { row - \@@_succ:n { #4 } }
4330   \dim_set_eq:NN \l_tmpc_dim \pgf@y
4331   \bool_lazy_and:nnT
4332     { \int_compare_p:nNn { #2 } > 1 }
4333     { ! \tl_if_blank_p:V \CT@drsc@ }
4334     {
4335       \group_begin:
4336       \CT@drsc@
4337       \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4338       \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
4339       \dim_set:Nn \l_tmpd_dim
4340         { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4341       \pgfpathrectanglecorners
4342         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4343         { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4344       \pgfusepath { fill }
4345       \group_end:
4346     }
4347   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4348   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4349   \prg_replicate:nn { #2 - 1 }
4350   {
4351     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4352     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4353     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4354     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4355   }
4356   \CT@arc@
4357   \pgfsetlinewidth { 1.1 \arrayrulewidth }
4358   \pgfsetrectcap
4359   \pgfusepathqstroke
4360 }

```

The following command draws a complete vertical rule in the column #1 (#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key `corners` is not used).

```

4361 \cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
4362 { \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

```

The command `\@@_draw_hlines:` 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).

```

4363 \cs_new_protected:Npn \@@_draw_vlines:
4364 {
4365   \int_step_inline:nnn
4366     { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
4367     { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
4368     {
4369       \tl_if_eq:NnF \l_@@_vlines_clist { all }
4370       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4371       { \@@_vline:nn { ##1 } 1 }
4372     }
4373 }

```

## The horizontal rules

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the row #1. #2 is the number of consecutive occurrences of `\Hline`.

```

4374 \cs_new_protected:Npn \@@_hline:nn #1 #2
4375 {
4376   \pgfpicture
4377   \@@_hline_i:nn { #1 } { #2 }
4378   \endpgfpicture
4379 }
4380 \cs_new_protected:Npn \@@_hline_i:nn #1 #2
4381 {

```

`\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_tmppc_tl`.

```

4382   \tl_set:Nn \l_tmpa_tl { #1 }
4383   \tl_clear_new:N \l_tmppc_tl
4384   \int_step_variable:nNn \c@jCol \l_tmpb_tl
4385   {

```

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.

```

4386     \bool_gset_true:N \g_tmpa_bool
4387     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4388     { \@@_test_hline_in_block:nnnn ##1 }
4389     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4390     { \@@_test_hline_in_block:nnnn ##1 }
4391     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4392     { \@@_test_hline_in_stroken_block:nnnn ##1 }
4393     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4394     \bool_if:NTF \g_tmpa_bool
4395     {
4396       \tl_if_empty:NT \l_tmppc_tl

```

We keep in memory that we have a rule to draw.

```

4397       { \tl_set_eq:NN \l_tmppc_tl \l_tmpb_tl }
4398     }
4399   {
4400     \tl_if_empty:NF \l_tmppc_tl
4401     {
4402       \@@_hline_ii:nnnn
4403       { #1 }
4404       { #2 }
4405       \l_tmppc_tl
4406       { \int_eval:n { \l_tmpb_tl - 1 } }
4407       \tl_clear:N \l_tmppc_tl
4408     }
4409   }
4410 }
4411 \tl_if_empty:NF \l_tmppc_tl
4412 {
4413   \@@_hline_ii:nnnn
4414   { #1 }
4415   { #2 }
4416   \l_tmppc_tl
4417   { \int_use:N \c@jCol }
4418   \tl_clear:N \l_tmppc_tl
4419 }
4420 }

```

```

4421 \cs_new_protected:Npn \@@_test_in_corner_h:
4422 {
4423   \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
4424   {
4425     \seq_if_in:NxT
4426     \l_@@_corners_cells_seq
4427     { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4428     { \bool_set_false:N \g_tmpa_bool }
4429   }
4430   {
4431     \seq_if_in:NxT
4432     \l_@@_corners_cells_seq
4433     { \l_tmpa_tl - \l_tmpb_tl }
4434     {
4435       \int_compare:nNnTF \l_tmpa_tl = 1
4436       { \bool_set_false:N \g_tmpa_bool }
4437       {
4438         \seq_if_in:NxT
4439         \l_@@_corners_cells_seq
4440         { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4441         { \bool_set_false:N \g_tmpa_bool }
4442       }
4443     }
4444   }
4445 }

```

#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.

```

4446 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
4447 {
4448   \pgfrememberpicturepositiononpagetrue
4449   \pgf@relevantforpicturesizefalse
4450   \@@_qpoint:n { col - #3 }
4451   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4452   \@@_qpoint:n { row - #1 }
4453   \dim_set_eq:NN \l_tmpb_dim \pgf@y
4454   \@@_qpoint:n { col - \@@_succ:n { #4 } }
4455   \dim_set_eq:NN \l_tmpc_dim \pgf@x
4456   \bool_lazy_and:nnT
4457   { \int_compare_p:nNn { #2 } > 1 }
4458   { ! \tl_if_blank_p:V \CT@drsc@ }
4459   {
4460     \group_begin:
4461     \CT@drsc@
4462     \dim_set:Nn \l_tmpd_dim
4463     { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4464     \pgfpathrectanglecorners
4465     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4466     { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4467     \pgfusepathqfill
4468     \group_end:
4469   }
4470   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4471   \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4472   \prg_replicate:nn { #2 - 1 }
4473   {
4474     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4475     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4476     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4477     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4478   }
4479   \CT@arc@
4480   \pgfsetlinewidth { 1.1 \arrayrulewidth }

```

```

4481 \pgfsetrectcap
4482 \pgfusepathqstroke
4483 }

4484 \cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
4485 { \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as `\Cdots` and in the corners (if the key `corners` is used).

```

4486 \cs_new_protected:Npn \@@_draw_hlines:
4487 {
4488   \int_step_inline:nnn
4489   { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
4490   { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
4491   {
4492     \tl_if_eq:NnF \l_@@_hlines_clist { all }
4493     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
4494     { \@@_hline:nn { ##1 } 1 }
4495   }
4496 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

4497 \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.

```

4498 \cs_set:Npn \@@_Hline_i:n #1
4499 {
4500   \peek_meaning_ignore_spaces:NTF \Hline
4501   { \@@_Hline_ii:nn { #1 + 1 } }
4502   { \@@_Hline_iii:n { #1 } }
4503 }

4504 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }

4505 \cs_set:Npn \@@_Hline_iii:n #1
4506 {
4507   \skip_vertical:n
4508   {
4509     \arrayrulewidth * ( #1 )
4510     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
4511   }
4512   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4513   { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
4514   \ifnum 0 = ` { \fi }
4515 }

```

## 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`.

```

4516 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4
4517 {
4518   \bool_lazy_all:nT
4519   {
4520     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
4521     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4522     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4523     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }

```



```

4524     }
4525     { \bool_gset_false:N \g_tmpa_bool }
4526 }

```

The same for vertical rules.

```

4527 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4
4528 {
4529     \bool_lazy_all:nT
4530     {
4531         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4532         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4533         { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
4534         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4535     }
4536     { \bool_gset_false:N \g_tmpa_bool }
4537 }

4538 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
4539 {
4540     \bool_lazy_all:nT
4541     {
4542         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4543         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
4544         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4545         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4546     }
4547     { \bool_gset_false:N \g_tmpa_bool }
4548 }

4549 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
4550 {
4551     \bool_lazy_all:nT
4552     {
4553         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4554         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4555         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4556         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
4557     }
4558     { \bool_gset_false:N \g_tmpa_bool }
4559 }

```

## 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.

```

4560 \cs_new_protected:Npn \@@_compute_corners:
4561 {

```

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.

```

4562     \seq_clear_new:N \l_@@_corners_cells_seq
4563     \clist_map_inline:Nn \l_@@_corners_clist
4564     {
4565         \str_case:nnF { ##1 }
4566         {
4567             { NW }
4568             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
4569             { NE }
4570             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
4571             { SW }
4572             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
4573             { SE }
4574             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }

```

```

4575     }
4576     { \@@_error:nn { bad~corner } { ##1 } }
4577 }

```

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.

```

4578 \seq_if_empty:NF \l_@@_corners_cells_seq
4579 {

```

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.

```

4580 \iow_now:Nn \@mainaux \ExplSyntaxOn
4581 \iow_now:Nx \@mainaux
4582 {
4583   \seq_gset_from_clist:cn
4584   { c_@@_corners_cells_ \int_use:N \g_@@_env_int _ seq }
4585   { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
4586 }
4587 \iow_now:Nn \@mainaux \ExplSyntaxOff
4588 }
4589 }

```

“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.

```

4590 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
4591 {

```

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.

```

4592 \bool_set_false:N \l_tmpa_bool
4593 \int_zero_new:N \l_@@_last_empty_row_int
4594 \int_set:Nn \l_@@_last_empty_row_int { #1 }
4595 \int_step_inline:nnnn { #1 } { #3 } { #5 }
4596 {
4597   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
4598   \bool_lazy_or:nnTF
4599   {
4600     \cs_if_exist_p:c
4601     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
4602   }
4603   \l_tmpb_bool
4604   { \bool_set_true:N \l_tmpa_bool }
4605   {
4606     \bool_if:NF \l_tmpa_bool
4607     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
4608   }
4609 }

```

Now, you determine the last empty cell in the row of number 1.

```

4610 \bool_set_false:N \l_tmpa_bool
4611 \int_zero_new:N \l_@@_last_empty_column_int
4612 \int_set:Nn \l_@@_last_empty_column_int { #2 }
4613 \int_step_inline:nnnn { #2 } { #4 } { #6 }
4614 {
4615   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
4616   \bool_lazy_or:nnTF
4617     \l_tmpb_bool
4618     {
4619       \cs_if_exist_p:c
4620         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
4621     }
4622     { \bool_set_true:N \l_tmpa_bool }
4623     {
4624       \bool_if:NF \l_tmpa_bool
4625       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
4626     }
4627 }

```

Now, we loop over the rows.

```

4628 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
4629 {

```

We treat the row number ##1 with another loop.

```

4630   \bool_set_false:N \l_tmpa_bool
4631   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
4632   {
4633     \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
4634     \bool_lazy_or:nnTF
4635       \l_tmpb_bool
4636       {
4637         \cs_if_exist_p:c
4638           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
4639       }
4640       { \bool_set_true:N \l_tmpa_bool }
4641       {
4642         \bool_if:NF \l_tmpa_bool
4643         {
4644           \int_set:Nn \l_@@_last_empty_column_int { #####1 }
4645           \seq_put_right:Nn
4646             \l_@@_corners_cells_seq
4647             { ##1 - #####1 }
4648         }
4649       }
4650   }
4651 }
4652 }

```

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`).

```

4653 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
4654 {
4655   \int_set:Nn \l_tmpa_int { #1 }
4656   \int_set:Nn \l_tmpb_int { #2 }
4657   \bool_set_false:N \l_tmpb_bool
4658   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4659     { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
4660 }
4661 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6
4662 {
4663   \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }

```

```

4664 {
4665   \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
4666   {
4667     \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
4668     {
4669       \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
4670       { \bool_set_true:N \l_tmpb_bool }
4671     }
4672   }
4673 }
4674 }

```

## 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`.

```

4675 \cs_new:Npn \@@_hdottedline:
4676 {
4677   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
4678   \@@_hdottedline_i:
4679 }

```

On the other side, the following command should be protected.

```

4680 \cs_new_protected:Npn \@@_hdottedline_i:
4681 {

```

We write in the code-after 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).

```

4682   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4683   { \@@_hdottedline:n { \int_use:N \c@iRow } }
4684 }

```

The command `\@@_hdottedline:n` is the command written in the `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```

4685 \AtBeginDocument
4686 {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That's why we construct now a version of `\@@_hdottedline:n` with the right environment (`\begin{pgfpicture}\end{pgfpicture}` or `\begin{tikzpicture}...\end{tikzpicture}`).

```

4687   \cs_new_protected:Npx \@@_hdottedline:n #1
4688   {
4689     \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
4689     \bool_set_true:N \exp_not:N \l_@@_final_open_bool
4691     \c_@@_pgfortikzpicture_tl
4692     \@@_hdottedline_i:n { #1 }
4693     \c_@@_endpgfortikzpicture_tl
4694   }
4695 }

```

The following command *must* be protected since it is used in the construction of `\@@_hdottedline:n`.

```

4696 \cs_new_protected:Npn \@@_hdottedline_i:n #1
4697 {
4698   \pgfrememberpicturepositiononpagetrue
4699   \@@_qpoint:n { row - #1 }

```

We do a translation par `-l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```

4700 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4701 \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
4702 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim

```

The dotted line will be extended if the user uses `margin` (or `left-margin` and `right-margin`).

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}

```

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

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}

```

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

```

4703 \@@_qpoint:n { col - 1 }
4704 \dim_set:Nn \l_@@_x_initial_dim
4705 {
4706 \pgf@x +

```

We do a reduction by `\arraycolsep` for the environments with delimiters (and not for the other).

```

4707 \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4708 - \l_@@_left_margin_dim
4709 }
4710 \@@_qpoint:n { col - \@@_succ:n \c@jCol }
4711 \dim_set:Nn \l_@@_x_final_dim
4712 {
4713 \pgf@x -
4714 \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4715 + \l_@@_right_margin_dim
4716 }

```

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.

```

4717 \tl_if_eq:NnF \g_@@_left_delim_tl (
4718 { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
4719 \tl_if_eq:NnF \g_@@_right_delim_tl )
4720 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }

```

Up to now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

4721 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4722 \@@_draw_line:
4723 }

```

## Vertical dotted lines

```

4724 \cs_new_protected:Npn \@@_vdottedline:n #1
4725 {
4726 \bool_set_true:N \l_@@_initial_open_bool
4727 \bool_set_true:N \l_@@_final_open_bool

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”.

```

4728   \bool_if:NTF \c_@@_tikz_loaded_bool
4729   {
4730     \tikzpicture
4731     \@@_vdottedline_i:n { #1 }
4732     \endtikzpicture
4733   }
4734   {
4735     \pgfpicture
4736     \@@_vdottedline_i:n { #1 }
4737     \endpgfpicture
4738   }
4739 }

```

```

4740 \cs_new_protected:Npn \@@_vdottedline_i:n #1
4741 {

```

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.

```

4742   \CT@arc@
4743   \pgfrememberpicturepositiononpagetrue
4744   \@@_qpoint:n { col - \int_eval:n { #1 + 1 } }

```

We do a translation `par -\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```

4745   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
4746   \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
4747   \@@_qpoint:n { row - 1 }

```

We arbitrary decrease the height of the dotted line by a quantity equal to `\l_@@_inter_dots_dim` in order to improve the visual impact.

```

4748   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
4749   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
4750   \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }

```

Up to now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

4751   \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4752   \@@_draw_line:
4753 }

```

## 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.

```

4754 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

4755 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
4756 {
4757   auto-columns-width .code:n =
4758   {
4759     \bool_set_true:N \l_@@_block_auto_columns_width_bool
4760     \dim_gzero_new:N \g_@@_max_cell_width_dim
4761     \bool_set_true:N \l_@@_auto_columns_width_bool
4762   }
4763 }

```

```

4764 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
4765 {
4766   \int_gincr:N \g_@@_NiceMatrixBlock_int
4767   \dim_zero:N \l_@@_columns_width_dim
4768   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
4769   \bool_if:NT \l_@@_block_auto_columns_width_bool
4770   {
4771     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4772     {
4773       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
4774       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4775     }
4776   }
4777 }

```

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).

```

4778 {
4779   \bool_if:NT \l_@@_block_auto_columns_width_bool
4780   {
4781     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
4782     \iow_shipout:Nx \@mainaux
4783     {
4784       \cs_gset:cpn
4785       { @@_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.

```

4786       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
4787     }
4788     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
4789   }
4790 }

```

## The extra nodes

First, two variants of the functions \dim\_min:nn and \dim\_max:nn.

```

4791 \cs_generate_variant:Nn \dim_min:nn { v n }
4792 \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).

```

4793 \cs_new_protected:Npn \@@_create_extra_nodes:
4794 {
4795   \bool_if:nTF \l_@@_medium_nodes_bool
4796   {
4797     \bool_if:NTF \l_@@_large_nodes_bool
4798     \@@_create_medium_and_large_nodes:
4799     \@@_create_medium_nodes:
4800   }
4801   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
4802 }

```

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`.

```

4803 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
4804 {
4805   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4806   {
4807     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
4808     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
4809     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
4810     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
4811   }
4812   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4813   {
4814     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
4815     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
4816     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
4817     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
4818   }

```

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

```

4819   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4820   {
4821     \int_step_variable:nnNn
4822     \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.

```

4823     {
4824       \cs_if_exist:cT
4825       { 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`.

```

4826     {
4827       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
4828       \dim_set:cn { l_@@_row_\@@_i: _min_dim }
4829       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
4830       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
4831       {
4832         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
4833         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
4834       }

```

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`.

```

4835       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
4836       \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
4837       { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
4838       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
4839       {
4840         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
4841         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
4842       }
4843     }
4844   }
4845 }

```



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

```

4846 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4847 {
4848   \dim_compare:nNnT
4849     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
4850     {
4851       \@@_qpoint:n { row - \@@_i: - base }
4852       \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
4853       \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
4854     }
4855 }
4856 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4857 {
4858   \dim_compare:nNnT
4859     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
4860     {
4861       \@@_qpoint:n { col - \@@_j: }
4862       \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
4863       \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
4864     }
4865 }
4866 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

4867 \cs_new_protected:Npn \@@_create_medium_nodes:
4868 {
4869   \pgfpicture
4870   \pgfrememberpicturepositiononpagetrue
4871   \pgf@relevantforpicturesizefalse
4872   \@@_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”.

```

4873   \tl_set:Nn \l_@@_suffix_tl { -medium }
4874   \@@_create_nodes:
4875   \endpgfpicture
4876 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones<sup>61</sup>. 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:`.

```

4877 \cs_new_protected:Npn \@@_create_large_nodes:
4878 {
4879   \pgfpicture
4880   \pgfrememberpicturepositiononpagetrue
4881   \pgf@relevantforpicturesizefalse
4882   \@@_computations_for_medium_nodes:
4883   \@@_computations_for_large_nodes:
4884   \tl_set:Nn \l_@@_suffix_tl { -large }
4885   \@@_create_nodes:
4886   \endpgfpicture
4887 }
4888 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
4889 {

```

---

<sup>61</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

4890 \pgfpicture
4891 \pgfrememberpicturepositiononpagetrue
4892 \pgf@relevantforpicturesizefalse
4893 \@@_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”.

```

4894 \tl_set:Nn \l_@@_suffix_tl { - medium }
4895 \@@_create_nodes:
4896 \@@_computations_for_large_nodes:
4897 \tl_set:Nn \l_@@_suffix_tl { - large }
4898 \@@_create_nodes:
4899 \endpgfpicture
4900 }

```

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.

```

4901 \cs_new_protected:Npn \@@_computations_for_large_nodes:
4902 {
4903 \int_set:Nn \l_@@_first_row_int 1
4904 \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`.

```

4905 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
4906 {
4907 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
4908 {
4909 (
4910 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
4911 \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4912 )
4913 / 2
4914 }
4915 \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4916 { l_@@_row _ \@@_i: _ min _ dim }
4917 }
4918 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
4919 {
4920 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
4921 {
4922 (
4923 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
4924 \dim_use:c
4925 { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4926 )
4927 / 2
4928 }
4929 \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4930 { l_@@_column _ \@@_j: _ max _ dim }
4931 }

```

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

```

4932 \dim_sub:cn
4933 { l_@@_column _ 1 _ min _ dim }
4934 \l_@@_left_margin_dim
4935 \dim_add:cn
4936 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
4937 \l_@@_right_margin_dim
4938 }

```

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  $\backslash l\_@@\_suffix\_tl$  (-medium or -large).

```

4939 \cs_new_protected:Npn \@@_create_nodes:
4940 {
4941   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4942   {
4943     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4944     {

```

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

```

4945       \@@_pgf_rect_node:nnnnn
4946       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4947       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
4948       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
4949       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
4950       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
4951       \str_if_empty:NF \l_@@_name_str
4952       {
4953         \pgfnodealias
4954         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4955         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4956       }
4957     }
4958   }

```

Now, we create the nodes for the cells of the  $\backslash multicolumn$ . We recall that we have stored in  $\backslash g\_@@\_multicolumn\_cells\_seq$  the list of the cells where a  $\backslash multicolumn\{n\}\{\dots\}\{\dots\}$  with  $n > 1$  was issued and in  $\backslash g\_@@\_multicolumn\_sizes\_seq$  the correspondent values of  $n$ .

```

4959   \seq_mapthread_function:NNN
4960   \g_@@_multicolumn_cells_seq
4961   \g_@@_multicolumn_sizes_seq
4962   \@@_node_for_multicolumn:nn
4963 }

```

```

4964 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
4965 {
4966   \cs_set_nopar:Npn \@@_i: { #1 }
4967   \cs_set_nopar:Npn \@@_j: { #2 }
4968 }

```

The command  $\backslash@@\_node\_for\_multicolumn:nn$  takes two arguments. The first is the position of the cell where the command  $\backslash multicolumn\{n\}\{\dots\}\{\dots\}$  was issued in the format  $i$ - $j$  and the second is the value of  $n$  (the length of the “multi-cell”).

```

4969 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
4970 {
4971   \@@_extract_coords_values: #1 \q_stop
4972   \@@_pgf_rect_node:nnnnn
4973   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4974   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
4975   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
4976   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
4977   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
4978   \str_if_empty:NF \l_@@_name_str
4979   {
4980     \pgfnodealias
4981     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4982     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
4983   }
4984 }

```

## 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.

```

4985 \keys_define:nn { NiceMatrix / Block / FirstPass }
4986 {
4987   l .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
4988   l .value_forbidden:n = true ,
4989   r .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
4990   r .value_forbidden:n = true ,
4991   c .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
4992   c .value_forbidden:n = true ,
4993   t .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl t ,
4994   t .value_forbidden:n = true ,
4995   b .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl b ,
4996   b .value_forbidden:n = true ,
4997   color .tl_set:N = \l_@@_color_tl ,
4998   color .value_required:n = true ,
4999 }

```

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 `>` (for TeX instructions put before the math mode of the label and before the beginning of the small array of the block). It's mandatory to use an expandable command.

```

5000 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
5001 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax  $i-j$ ) has not been provided by the user, you use `1-1` (that is to say a block of only one cell).

```

5002   \peek_remove_spaces:n
5003   {
5004     \tl_if_blank:nTF { #2 }
5005     { \@@_Block_i 1-1 \q_stop }
5006     { \@@_Block_i #2 \q_stop }
5007     { #1 } { #3 } { #4 }
5008   }
5009 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

5010 \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.

```

5011 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5012 {

```

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).

```

5013   \bool_lazy_or:nnTF
5014   { \tl_if_blank_p:n { #1 } }
5015   { \str_if_eq_p:nn { #1 } { * } }
5016   { \int_set:Nn \l_tmpa_int { 100 } }
5017   { \int_set:Nn \l_tmpa_int { #1 } }
5018   \bool_lazy_or:nnTF

```

```

5019 { \tl_if_blank_p:n { #2 } }
5020 { \str_if_eq_p:nn { #2 } { * } }
5021 { \int_set:Nn \l_tmpb_int { 100 } }
5022 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5023 \int_compare:nNnTF \l_tmpb_int = 1
5024 {
5025   \tl_if_empty:NTF \l_@@_cell_type_tl
5026   { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5027   { \tl_set_eq:NN \l_@@_hpos_of_block_tl \l_@@_cell_type_tl }
5028 }
5029 { \tl_set:Nn \l_@@_hpos_of_block_tl c }

```

The value of `\l_@@_hpos_of_block_tl` may be modified by the keys of the command `\Block` that we will analyze now.

```

5030 \keys_set:known:n { NiceMatrix / Block / FirstPass } { #3 }
5031 \tl_set:Nx \l_tmpa_tl
5032 {
5033   { \int_use:N \c@iRow }
5034   { \int_use:N \c@jCol }
5035   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5036   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5037 }

```

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).

```

5038 \bool_lazy_or:nnTF
5039 { \int_compare_p:nNn { \l_tmpa_int } = 1 }
5040 { \int_compare_p:nNn { \l_tmpb_int } = 1 }
5041 { \exp_args:Nxx \@@_Block_iv:nnnnn }
5042 { \exp_args:Nxx \@@_Block_v:nnnnn }
5043 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5044 }

```

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).

```

5045 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5046 {
5047   \int_gincr:N \g_@@_block_box_int
5048   \set_protected_nopar:Npn \diagbox ##1 ##2
5049   {
5050     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5051     {
5052       \@@_actually_diagbox:nnnnnn
5053       { \int_use:N \c@iRow }
5054       { \int_use:N \c@jCol }
5055       { \int_eval:n { \c@iRow + #1 - 1 } }
5056       { \int_eval:n { \c@jCol + #2 - 1 } }
5057       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5058     }
5059   }
5060   \box_gclear_new:c
5061   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }

```

```

5062 \hbox_gset:cn
5063 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5064 {

```

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`: because that command seems to be bugged: it doesn't work in XeLaTeX when `fontspec` is loaded.

```

5065 \tl_if_empty:NTF \l_@@_color_tl
5066 { \int_compare:nNnT { #2 } = 1 \set@color }
5067 { \color { \l_@@_color_tl } }
5068 \group_begin:
5069 \cs_set:Npn \arraystretch { 1 }
5070 \dim_set_eq:NN \extrarowheight \c_zero_dim
5071 #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).

```

5072 \bool_if:NT \g_@@_rotate_bool { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5073 \bool_if:NTF \l_@@_NiceTabular_bool
5074 {
5075   \use:x
5076   {
5077     \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5078     { @ { } \l_@@_hpos_of_block_tl @ { } }
5079   }
5080   #5
5081   \end { tabular }
5082 }
5083 {
5084   \c_math_toggle_token
5085   \use:x
5086   {
5087     \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5088     { @ { } \l_@@_hpos_of_block_tl @ { } }
5089   }
5090   #5
5091   \end { array }
5092   \c_math_toggle_token
5093 }
5094 \group_end:
5095 }
5096 \bool_if:NT \g_@@_rotate_bool
5097 {
5098   \box_grotate:cn
5099   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5100   { 90 }
5101   \bool_gset_false:N \g_@@_rotate_bool
5102 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5103 \int_compare:nNnT { #2 } = 1
5104 {
5105   \dim_gset:Nn \g_@@_blocks_wd_dim
5106   {
5107     \dim_max:nn
5108     \g_@@_blocks_wd_dim
5109     {
5110       \box_wd:c
5111       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

```

5112     }
5113   }
5114 }

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.

5115   \int_compare:nNnT { #1 } = 1
5116   {
5117     \dim_gset:Nn \g_@@_blocks_ht_dim
5118     {
5119       \dim_max:nn
5120       \g_@@_blocks_ht_dim
5121       {
5122         \box_ht:c
5123         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5124       }
5125     }
5126     \dim_gset:Nn \g_@@_blocks_dp_dim
5127     {
5128       \dim_max:nn
5129       \g_@@_blocks_dp_dim
5130       {
5131         \box_dp:c
5132         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5133       }
5134     }
5135   }
5136   \seq_gput_right:Nx \g_@@_blocks_seq
5137   {
5138     \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\_of\_block\_tl. 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\_of\_block\_tl, which is fixed by the type of current column.

```

5139     { \exp_not:n { #3 } , \l_@@_hpos_of_block_tl }
5140     {
5141       \box_use_drop:c
5142       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5143     }
5144   }
5145 }

```

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.

```

5146 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5147 {
5148   \seq_gput_right:Nx \g_@@_blocks_seq
5149   {
5150     \l_tmpa_tl
5151     { \exp_not:n { #3 } }
5152     \exp_not:n
5153     {
5154       {
5155         \bool_if:NTF \l_@@_NiceTabular_bool
5156         {
5157           \group_begin:
5158           \cs_set:Npn \arraystretch { 1 }
5159           \dim_set_eq:NN \extrarowheight \c_zero_dim
5160           #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).

```

5161         \bool_if:NT \g_@@_rotate_bool
5162         { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5163     \use:x
5164     {
5165         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5166         { @ { } \l_@@_hpos_of_block_tl @ { } }
5167     }
5168     #5
5169     \end { tabular }
5170     \group_end:
5171 }
5172 {
5173     \group_begin:
5174     \cs_set:Npn \arraystretch { 1 }
5175     \dim_set_eq:NN \extrarowheight \c_zero_dim
5176     #4
5177     \bool_if:NT \g_@@_rotate_bool
5178     { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5179     \c_math_toggle_token
5180     \use:x
5181     {
5182         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5183         { @ { } \l_@@_hpos_of_block_tl @ { } }
5184     }
5185     #5
5186     \end { array }
5187     \c_math_toggle_token
5188     \group_end:
5189 }
5190 }
5191 }
5192 }
5193 }

```

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).

```

5194 \keys_define:nn { NiceMatrix / Block / SecondPass }
5195 {
5196     fill .tl_set:N = \l_@@_fill_tl ,
5197     fill .value_required:n = true ,
5198     draw .tl_set:N = \l_@@_draw_tl ,
5199     draw .default:n = default ,
5200     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5201     rounded-corners .default:n = 4 pt ,
5202     color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5203     color .value_required:n = true ,
5204     borders .clist_set:N = \l_@@_borders_clist ,
5205     borders .value_required:n = true ,
5206     hvlines .bool_set:N = \l_@@_hvlines_block_bool ,
5207     hvlines .default:n = true ,
5208     line-width .dim_set:N = \l_@@_line_width_dim ,
5209     line-width .value_required:n = true ,
5210     l .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
5211     l .value_forbidden:n = true ,
5212     r .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
5213     r .value_forbidden:n = true ,
5214     c .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
5215     c .value_forbidden:n = true ,

```



```

5216   t .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl t ,
5217   t .value_forbidden:n = true ,
5218   b .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl b ,
5219   b .value_forbidden:n = true ,
5220   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
5221 }

```

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.

```

5222 \cs_new_protected:Npn \@@_draw_blocks:
5223 {
5224   \cs_set_eq:NN \ialign \@@_old_ialign:
5225   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
5226 }
5227 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5228 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5229   \int_zero_new:N \l_@@_last_row_int
5230   \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.

```

5231   \int_compare:nNnTF { #3 } > { 99 }
5232   { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
5233     { \int_set:Nn \l_@@_last_row_int { #3 } }
5234   \int_compare:nNnTF { #4 } > { 99 }
5235   { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
5236     { \int_set:Nn \l_@@_last_col_int { #4 } }
5237   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
5238   {
5239     \int_compare:nTF
5240     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
5241     {
5242       \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
5243       \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
5244       \group_begin:
5245       \globaldefs = 1
5246       \@@_msg_redirect_name:nn { columns~not~used } { none }
5247       \group_end:
5248     }
5249     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5250   }
5251   {
5252     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
5253     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5254     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
5255   }
5256 }
5257 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
5258 {

```

The sequence of the positions of the blocks will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

5259   \seq_gput_left:Nn \g_@@_pos_of_blocks_seq { { #1 } { #2 } { #3 } { #4 } }

```

The group is for the keys.

```

5260 \group_begin:
5261 \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
5262 \tl_if_empty:NF \l_@@_draw_tl
5263 {
5264   \tl_gput_right:Nx \g_nicematrix_code_after_tl
5265   {
5266     \@@_stroke_block:nnn
5267     { \exp_not:n { #5 } }
5268     { #1 - #2 }
5269     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5270   }
5271   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
5272   { { #1 } { #2 } { #3 } { #4 } }
5273 }
5274 \bool_if:NT \l_@@_hvlines_block_bool
5275 {
5276   \tl_gput_right:Nx \g_nicematrix_code_after_tl
5277   {
5278     \@@_hvlines_block:nnn
5279     { \exp_not:n { #5 } }
5280     { #1 - #2 }
5281     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5282   }
5283 }
5284 \clist_if_empty:NF \l_@@_borders_clist
5285 {
5286   \tl_gput_right:Nx \g_nicematrix_code_after_tl
5287   {
5288     \@@_stroke_borders_block:nnn
5289     { \exp_not:n { #5 } }
5290     { #1 - #2 }
5291     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5292   }
5293 }
5294 \tl_if_empty:NF \l_@@_fill_tl
5295 {

```

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`.

```

5296   \exp_last_unbraced:NV \@@_extract_brackets \l_@@_fill_tl \q_stop
5297   \tl_gput_right:Nx \g_nicematrix_code_before_tl
5298   {
5299     \exp_not:N \roundedrectanglecolor
5300     [ \l_tmpa_tl ]
5301     { \exp_not:V \l_tmpb_tl }
5302     { #1 - #2 }
5303     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5304     { \dim_use:N \l_@@_rounded_corners_dim }
5305   }
5306 }
5307 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5308 {
5309   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5310   {
5311     \@@_actually_diagbox:nnnnnn
5312     { #1 }
5313     { #2 }
5314     { \int_use:N \l_@@_last_row_int }
5315     { \int_use:N \l_@@_last_col_int }
5316     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }

```

```

5317         }
5318     }

5319     \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
5320     \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`. The latter will be used by `nicematrix` to put the label of the node. The first one won't be used explicitly.

```

\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.

```

5321 \pgfpicture
5322 \pgfrememberpicturepositiononpagetrue
5323 \pgf@relevantforpicturesizefalse
5324 \@@_qpoint:n { row - #1 }
5325 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5326 \@@_qpoint:n { col - #2 }
5327 \dim_set_eq:NN \l_tmpb_dim \pgf@x
5328 \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
5329 \dim_set_eq:NN \l_tmpc_dim \pgf@y
5330 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5331 \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.

```

5332 \begin { pgfscope }
5333 \@@_pgf_rect_node:nnnnn
5334 { \@@_env: - #1 - #2 - block }
5335 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
5336 \end { pgfscope }

```

We construct the short node.

```

5337 \dim_set_eq:NN \l_tmpb_dim \c_max_dim
5338 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5339 {

```

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.

```

5340 \cs_if_exist:cT
5341 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5342 {
5343 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5344 {
5345 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
5346 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }

```

```

5347     }
5348   }
5349 }

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.

5350   \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
5351   {
5352     \@@_qpoint:n { col - #2 }
5353     \dim_set_eq:NN \l_tmpb_dim \pgf@x
5354   }
5355   \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
5356   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5357   {
5358     \cs_if_exist:cT
5359     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5360     {
5361       \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5362       {
5363         \pgfpointanchor
5364         { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5365         { east }
5366         \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
5367       }
5368     }
5369   }
5370   \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
5371   {
5372     \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5373     \dim_set_eq:NN \l_tmpd_dim \pgf@x
5374   }
5375   \@@_pgf_rect_node:nnnnn
5376   { \@@_env: - #1 - #2 - block - short }
5377   \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpe_dim

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@_pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```

5378   \bool_if:NT \l_@@_medium_nodes_bool
5379   {
5380     \@@_pgf_rect_node:nnn
5381     { \@@_env: - #1 - #2 - block - medium }
5382     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
5383     {
5384       \pgfpointanchor
5385       { \@@_env:
5386         - \int_use:N \l_@@_last_row_int
5387         - \int_use:N \l_@@_last_col_int - medium
5388       }
5389       { south-east }
5390     }
5391   }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

5392   \int_compare:nNnTF { #1 } = { #3 }
5393   {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

5394     \int_compare:nNnTF { #1 } = 0
5395     { \l_@@_code_for_first_row_tl }
5396     {
5397       \int_compare:nNnT { #1 } = \l_@@_last_row_int
5398       \l_@@_code_for_last_row_tl
5399     }

```

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`.

```
5400 \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }
```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```
5401 \pgfpointanchor
5402 { \@@_env: - #1 - #2 - block - short }
5403 {
5404   \str_case:Vn \l_@@_hpos_of_block_tl
5405   {
5406     c { center }
5407     l { west }
5408     r { east }
5409   }
5410 }
```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```
5411 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
5412 \pgfset { inner~sep = \c_zero_dim }
5413 \pgfnode
5414 { rectangle }
5415 {
5416   \str_case:Vn \l_@@_hpos_of_block_tl
5417   {
5418     c { base }
5419     l { base~west }
5420     r { base~east }
5421   }
5422 }
5423 { \box_use_drop:N \l_@@_cell_box } { } { }
5424 }
```

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`).

```
5425 {
```

If we are in the first column, we must put the block as if it was with the key `r`.

```
5426 \int_compare:nNnT { #2 } = 0
5427 { \tl_set:Nn \l_@@_hpos_of_block_tl r }
5428 \bool_if:nT \g_@@_last_col_found_bool
5429 {
5430   \int_compare:nNnT { #2 } = \g_@@_col_total_int
5431   { \tl_set:Nn \l_@@_hpos_of_block_tl l }
5432 }
5433 \pgftransformshift
5434 {
5435   \pgfpointanchor
5436   { \@@_env: - #1 - #2 - block - short }
5437   {
5438     \str_case:Vn \l_@@_hpos_of_block_tl
5439     {
5440       c { center }
5441       l { west }
5442       r { east }
5443     }
5444   }
5445 }
5446 \pgfset { inner~sep = \c_zero_dim }
5447 \pgfnode
5448 { rectangle }
5449 {
5450   \str_case:Vn \l_@@_hpos_of_block_tl
5451   {
```

```

5452         c { center }
5453         l { west }
5454         r { east }
5455     }
5456 }
5457 { \box_use_drop:N \l_@@_cell_box } { } { }
5458 }
5459 \endpgfpicture
5460 \group_end:
5461 }

5462 \NewDocumentCommand \@@_extract_brackets { 0 { } }
5463 {
5464     \tl_set:Nn \l_tmpa_tl { #1 }
5465     \@@_store_in_tmpb_tl
5466 }
5467 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
5468 { \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).

```

5469 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
5470 {
5471     \group_begin:
5472     \tl_clear:N \l_@@_draw_tl
5473     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5474     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
5475     \pgfpicture
5476     \pgfrememberpicturepositiononpagetrue
5477     \pgf@relevantforpicturesizefalse
5478     \tl_if_empty:NF \l_@@_draw_tl
5479     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

5480         \str_if_eq:VnTF \l_@@_draw_tl { default }
5481         { \CT@arc@ }
5482         { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
5483     }
5484     \pgfsetcornersarced
5485     {
5486         \pgfpoint
5487         { \dim_use:N \l_@@_rounded_corners_dim }
5488         { \dim_use:N \l_@@_rounded_corners_dim }
5489     }
5490     \@@_cut_on_hyphen:w #2 \q_stop
5491     \bool_lazy_and:nnT
5492     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
5493     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
5494     {
5495         \@@_qpoint:n { row - \l_tmpa_tl }
5496         \dim_set:Nn \l_tmpb_dim { \pgf@y }
5497         \@@_qpoint:n { col - \l_tmpb_tl }
5498         \dim_set:Nn \l_tmpc_dim { \pgf@x }
5499         \@@_cut_on_hyphen:w #3 \q_stop
5500         \int_compare:nNnT \l_tmpa_tl > \c@iRow
5501         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
5502         \int_compare:nNnT \l_tmpb_tl > \c@jCol
5503         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5504         \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
5505         \dim_set:Nn \l_tmpa_dim { \pgf@y }

```

```

5506 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
5507 \dim_set:Nn \l_tmpd_dim { \pgf@x }
5508 \pgfpathrectanglecorners
5509 { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5510 { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5511 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

5512 \pgfusepath { stroke }
5513 }
5514 \endpgfpicture
5515 \group_end:
5516 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

5517 \keys_define:nn { NiceMatrix / BlockStroke }
5518 {
5519   color .tl_set:N = \l_@@_draw_tl ,
5520   draw .tl_set:N = \l_@@_draw_tl ,
5521   draw .default:n = default ,
5522   line-width .dim_set:N = \l_@@_line_width_dim ,
5523   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5524   rounded-corners .default:n = 4 pt
5525 }

```

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).

```

5526 \cs_new_protected:Npn \@@_hvlines_block:nnn #1 #2 #3
5527 {
5528   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5529   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5530   \@@_cut_on_hyphen:w #2 \q_stop
5531   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5532   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5533   \@@_cut_on_hyphen:w #3 \q_stop
5534   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5535   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5536   \pgfpicture
5537   \pgfrememberpicturepositiononpagetrue
5538   \pgf@relevantforpicturesizefalse
5539   \CT@arc@
5540   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

First, the vertical rules.

```

5541 \@@_qpoint:n { row - \l_tmpa_tl }
5542 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5543 \@@_qpoint:n { row - \l_tmpc_tl }
5544 \dim_set_eq:NN \l_tmpb_dim \pgf@y
5545 \int_step_inline:nnn \l_tmpd_tl \l_tmpb_tl
5546 {
5547   \@@_qpoint:n { col - ##1 }
5548   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpa_dim }
5549   \pgfpathlineto { \pgfpoint \pgf@x \l_tmpb_dim }
5550   \pgfusepathqstroke
5551 }

```

Now, the horizontal rules.

```

5552 \@@_qpoint:n { col - \l_tmpb_tl }
5553 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
5554 \@@_qpoint:n { col - \l_tmpd_tl }
5555 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \arrayrulewidth }
5556 \int_step_inline:nnn \l_tmpc_tl \l_tmpa_tl
5557 {

```

```

5558     \@@_qpoint:n { row - ##1 }
5559     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5560     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5561     \pgfusepathqstroke
5562   }
5563   \endpgfpicture
5564 }

```

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).

```

5565 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
5566 {
5567   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5568   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5569   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
5570   { \@@_error:n { borders~forbidden } }
5571   {
5572     \clist_map_inline:Nn \l_@@_borders_clist
5573     {
5574       \clist_if_in:nnF { top , bottom , left , right } { ##1 }
5575       { \@@_error:nn { bad-border } { ##1 } }
5576     }
5577     \@@_cut_on_hyphen:w #2 \q_stop
5578     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5579     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5580     \@@_cut_on_hyphen:w #3 \q_stop
5581     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5582     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5583     \pgfpicture
5584     \pgfrememberpicturepositiononpagetrue
5585     \pgf@relevantforpicturesizefalse
5586     \CT@arc@
5587     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
5588     \clist_if_in:NnT \l_@@_borders_clist { right }
5589     { \@@_stroke_vertical:n \l_tmpb_tl }
5590     \clist_if_in:NnT \l_@@_borders_clist { left }
5591     { \@@_stroke_vertical:n \l_tmpd_tl }
5592     \clist_if_in:NnT \l_@@_borders_clist { bottom }
5593     { \@@_stroke_horizontal:n \l_tmpa_tl }
5594     \clist_if_in:NnT \l_@@_borders_clist { top }
5595     { \@@_stroke_horizontal:n \l_tmpc_tl }
5596     \endpgfpicture
5597   }
5598 }

```

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).

```

5599 \cs_new_protected:Npn \@@_stroke_vertical:n #1
5600 {
5601   \@@_qpoint:n \l_tmpc_tl
5602   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
5603   \@@_qpoint:n \l_tmpa_tl
5604   \dim_set:Nn \l_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
5605   \@@_qpoint:n { #1 }
5606   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
5607   \pgfpathlineto { \pgfpoint \pgf@x \l_tmpc_dim }
5608   \pgfusepathqstroke
5609 }

```

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).



```

5610 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
5611 {
5612   \@@_qpoint:n \l_tmpd_tl
5613   \clist_if_in:NnTF \l_@@_borders_clist { left }
5614     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
5615     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
5616   \@@_qpoint:n \l_tmpb_tl
5617   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
5618   \@@_qpoint:n { #1 }
5619   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5620   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5621   \pgfusepathqstroke
5622 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

5623 \keys_define:nn { NiceMatrix / BlockBorders }
5624 {
5625   borders .clist_set:N = \l_@@_borders_clist ,
5626   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5627   rounded-corners .default:n = 4 pt ,
5628   line-width .dim_set:N = \l_@@_line_width_dim
5629 }

```

## How to draw the dotted lines transparently

```

5630 \cs_set_protected:Npn \@@_renew_matrix:
5631 {
5632   \RenewDocumentEnvironment { pmatrix } { } { }
5633   { \pNiceMatrix }
5634   { \endpNiceMatrix }
5635   \RenewDocumentEnvironment { vmatrix } { } { }
5636   { \vNiceMatrix }
5637   { \endvNiceMatrix }
5638   \RenewDocumentEnvironment { Vmatrix } { } { }
5639   { \VNiceMatrix }
5640   { \endVNiceMatrix }
5641   \RenewDocumentEnvironment { bmatrix } { } { }
5642   { \bNiceMatrix }
5643   { \endbNiceMatrix }
5644   \RenewDocumentEnvironment { Bmatrix } { } { }
5645   { \BNiceMatrix }
5646   { \endBNiceMatrix }
5647 }

```

## Automatic arrays

```

5648 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
5649 {
5650   \int_set:Nn \l_@@_nb_rows_int { #1 }
5651   \int_set:Nn \l_@@_nb_cols_int { #2 }
5652 }
5653 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
5654 {
5655   \int_zero_new:N \l_@@_nb_rows_int
5656   \int_zero_new:N \l_@@_nb_cols_int
5657   \@@_set_size:n #4 \q_stop
5658   \begin { NiceArrayWithDelims } { #1 } { #2 }
5659     { * { \l_@@_nb_cols_int } { c } } [ #3 , #5 , #7 ]
5660   \int_compare:nNnT \l_@@_first_row_int = 0
5661     {
5662       \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

```

5663     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5664     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5665   }
5666   \prg_replicate:nn \l_@@_nb_rows_int
5667   {
5668     \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

You 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`).

```

5669     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
5670     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5671   }
5672   \int_compare:nNnT \l_@@_last_row_int > { -2 }
5673   {
5674     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5675     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5676     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5677   }
5678   \end { NiceArrayWithDelims }
5679 }
5680 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
5681 {
5682   \cs_set_protected:cpn { #1 AutoNiceMatrix }
5683   {
5684     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
5685     \AutoNiceMatrixWithDelims { #2 } { #3 }
5686   }
5687 }
5688 \@@_define_com:nnn p ( )
5689 \@@_define_com:nnn b [ ]
5690 \@@_define_com:nnn v | |
5691 \@@_define_com:nnn V \l \l
5692 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

5693 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
5694 {
5695   \group_begin:
5696   \bool_set_true:N \l_@@_NiceArray_bool
5697   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
5698   \group_end:
5699 }

```

## The redefinition of the command `\dotfill`

```

5700 \cs_set_eq:NN \@@_old_dotfill \dotfill
5701 \cs_new_protected:Npn \@@_dotfill:
5702 {

```

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}`).

```

5703   \@@_old_dotfill
5704   \bool_if:NT \l_@@_NiceTabular_bool
5705   { \group_insert_after:N \@@_dotfill_ii: }
5706   { \group_insert_after:N \@@_dotfill_i: }
5707 }
5708 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
5709 \cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

5710 \cs_new_protected:Npn \@@_dotfill_iii:
5711 { \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`.

```

5712 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
5713 {
5714   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5715   {
5716     \@@_actually_diagbox:nnnnnn
5717     { \int_use:N \c@iRow }
5718     { \int_use:N \c@jCol }
5719     { \int_use:N \c@iRow }
5720     { \int_use:N \c@jCol }
5721     { \exp_not:n { #1 } }
5722     { \exp_not:n { #2 } }
5723   }

```

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`.

```

5724   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
5725   {
5726     { \int_use:N \c@iRow }
5727     { \int_use:N \c@jCol }
5728     { \int_use:N \c@iRow }
5729     { \int_use:N \c@jCol }
5730   }
5731 }

```

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 two other are the elements to draw below and above the diagonal line.

```

5732 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
5733 {
5734   \pgfpicture
5735   \pgf@relevantforpicturesizefalse
5736   \pgfrememberpicturepositiononpagetrue
5737   \@@_qpoint:n { row - #1 }
5738   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5739   \@@_qpoint:n { col - #2 }
5740   \dim_set_eq:NN \l_tmpb_dim \pgf@x
5741   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5742   \@@_qpoint:n { row - \@@_succ:n { #3 } }
5743   \dim_set_eq:NN \l_tmpc_dim \pgf@y
5744   \@@_qpoint:n { col - \@@_succ:n { #4 } }
5745   \dim_set_eq:NN \l_tmpd_dim \pgf@x
5746   \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
5747   {

```

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.

```

5748   \CT@arc@
5749   \pgfsetroundcap
5750   \pgfusepathqstroke
5751 }
5752 \pgfset { inner~sep = 1 pt }
5753 \pgfscope
5754 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
5755 \pgfnode { rectangle } { south-west }
5756 { \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }

```

```

5757 \endpgfscope
5758 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5759 \pgfnode { rectangle } { north-east }
5760 { \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
5761 \endpgfpicture
5762 }

```

## 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.

```

5763 \keys_define:nn { NiceMatrix }
5764 {
5765   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
5766   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
5767 }
5768 \keys_define:nn { NiceMatrix / CodeAfter }
5769 {
5770   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
5771   sub-matrix .value_required:n = true ,
5772   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
5773   delimiters / color .value_required:n = true ,
5774   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
5775   rules .value_required:n = true ,
5776   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
5777 }

```

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. 101.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

5778 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_i:n }

```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_i:n` which do *not* begin with `\omit` (and thus, the user will be able to use `\CodeAfter` without error and without the need to prefix by `\omit`).

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

5779 \cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
5780 {
5781   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
5782   \@@_CodeAfter_ii:n
5783 }

```

We catch the argument of the command `\end` (in `#1`).

```

5784 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1
5785 {

```

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.

```

5786   \str_if_eq:eeTF \@@_currenenv { #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`.

```

5787   {
5788     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
5789     \@@_CodeAfter_i:n
5790   }
5791 }

```

## The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_internal_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```

5792 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
5793 {
5794   \pgfpicture
5795   \pgfrememberpicturepositiononpagetrue
5796   \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.

```

5797   \@@_qpoint:n { row - 1 }
5798   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5799   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
5800   \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).

```

5801   \bool_if:nTF { #3 }
5802   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
5803   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
5804   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5805   {
5806     \cs_if_exist:cT
5807     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5808     {
5809       \pgfpointanchor
5810       { \@@_env: - ##1 - #2 }
5811       { \bool_if:nTF { #3 } { west } { east } }
5812       \dim_set:Nn \l_tmpa_dim
5813       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
5814     }
5815   }

```

Now we can put the delimiter with a node of PGF.

```

5816   \pgfset { inner~sep = \c_zero_dim }
5817   \dim_zero:N \nulldelimiterspace
5818   \pgftransformshift
5819   {
5820     \pgfpoint
5821     { \l_tmpa_dim }
5822     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
5823   }
5824   \pgfnode
5825   { rectangle }
5826   { \bool_if:nTF { #3 } { east } { west } }
5827   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

5828   \nullfont
5829   \c_math_toggle_token
5830   \tl_if_empty:NF \l_@@_delimiters_color_tl
5831   { \color { \l_@@_delimiters_color_tl } }
5832   \bool_if:nTF { #3 } { \left #1 } { \left . }

```

```

5833     \vcenter
5834     {
5835         \nullfont
5836         \hrule \@height
5837             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
5838             \@depth \c_zero_dim
5839             \@width \c_zero_dim
5840     }
5841     \bool_if:nTF { #3 } { \right . } { \right #1 }
5842     \c_math_toggle_token
5843 }
5844 { }
5845 { }
5846 \endpgfpicture
5847 }

```

## The command `\SubMatrix`

```

5848 \keys_define:nn { NiceMatrix / sub-matrix }
5849 {
5850     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
5851     extra-height .value_required:n = true ,
5852     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
5853     left-xshift .value_required:n = true ,
5854     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
5855     right-xshift .value_required:n = true ,
5856     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
5857     xshift .value_required:n = true ,
5858     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
5859     delimiters / color .value_required:n = true ,
5860     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
5861     slim .default:n = true ,
5862     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
5863     hlines .default:n = all ,
5864     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
5865     vlines .default:n = all ,
5866     hvlines .meta:n = { hlines, vlines } ,
5867     hvlines .value_forbidden:n = true ,
5868 }
5869 \keys_define:nn { NiceMatrix }
5870 {
5871     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
5872     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5873     NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5874     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5875     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5876     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5877 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

5878 \keys_define:nn { NiceMatrix / SubMatrix }
5879 {
5880     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
5881     hlines .default:n = all ,
5882     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
5883     vlines .default:n = all ,
5884     hvlines .meta:n = { hlines, vlines } ,
5885     hvlines .value_forbidden:n = true ,
5886     name .code:n =
5887         \tl_if_empty:nTF { #1 }
5888         { \@@_error:n { Invalid-name-format } }
5889         {

```

```

5890 \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
5891 {
5892   \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
5893   { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
5894   {
5895     \str_set:Nn \l_@@_submatrix_name_str { #1 }
5896     \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
5897   }
5898 }
5899 { \@@_error:n { Invalid-name-format } }
5900 } ,
5901 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
5902 rules .value_required:n = true ,
5903 code .tl_set:N = \l_@@_code_tl ,
5904 code .value_required:n = true ,
5905 name .value_required:n = true ,
5906 unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
5907 }

5908 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
5909 {
5910   \peek_remove_spaces:n
5911   {
5912     \@@_cut_on_hyphen:w #3 \q_stop
5913     \tl_clear_new:N \l_tmpc_tl
5914     \tl_clear_new:N \l_tmpd_tl
5915     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5916     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5917     \@@_cut_on_hyphen:w #2 \q_stop
5918     \seq_gput_right:Nx \g_@@_submatrix_seq
5919     { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
5920     \tl_gput_right:Nn \g_@@_internal_code_after_tl
5921     { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
5922   }
5923 }

```

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.

```

5924 \NewDocumentCommand \@@_SubMatrix { m m m m O { } }
5925 {
5926   \peek_remove_spaces:n
5927   { \@@_sub_matrix:nnnnn { #1 } { #2 } { #3 } { #4 } { #5 } }
5928 }
5929 \cs_new_protected:Npn \@@_sub_matrix:nnnnn #1 #2 #3 #4 #5
5930 {
5931   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

5932 \tl_clear_new:N \l_@@_first_i_tl
5933 \tl_clear_new:N \l_@@_first_j_tl
5934 \tl_clear_new:N \l_@@_last_i_tl
5935 \tl_clear_new:N \l_@@_last_j_tl

```

The command `\@@_cut_on_hyphen:w` cuts on the hyphen an argument of the form  $i-j$ . The value of  $i$  is stored in `\l_tmpa_tl` and the value of  $j$  is stored in `\l_tmpb_tl`.

```

5936 \@@_cut_on_hyphen:w #2 \q_stop
5937 \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
5938 \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
5939 \@@_cut_on_hyphen:w #3 \q_stop
5940 \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
5941 \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
5942 \bool_lazy_or:nnTF
5943 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
5944 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
5945 { \@@_error:n { SubMatrix~too~large } }
5946 {
5947   \str_clear_new:N \l_@@_submatrix_name_str
5948   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
5949   \pgfpicture
5950   \pgfrememberpicturepositiononpagetrue
5951   \pgf@relevantforpicturesizefalse
5952   \pgfset { inner-sep = \c_zero_dim }
5953   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
5954   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

5955 \bool_if:NTF \l_@@_submatrix_slim_bool
5956 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
5957 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
5958 {
5959   \cs_if_exist:cT
5960   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
5961   {
5962     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
5963     \dim_set:Nn \l_@@_x_initial_dim
5964     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
5965   }
5966   \cs_if_exist:cT
5967   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
5968   {
5969     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
5970     \dim_set:Nn \l_@@_x_final_dim
5971     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
5972   }
5973 }
5974 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
5975 { \@@_error:nn { impossible-delimiter } { left } }
5976 {
5977   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
5978   { \@@_error:nn { impossible~delimiter } { right } }
5979   { \@@_sub_matrix_i:nn { #1 } { #4 } }
5980 }
5981 \endpgfpicture
5982 }
5983 \group_end:
5984 }

```

#1 is the left delimiter dans #2 is the right one.

```

5985 \cs_new_protected:Npn \@@_sub_matrix_i:nn #1 #2
5986 {
5987   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
5988   \dim_set:Nn \l_@@_y_initial_dim
5989   { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
5990   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
5991   \dim_set:Nn \l_@@_y_final_dim
5992   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }

```



```

5993 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
5994 {
5995   \cs_if_exist:cT
5996   { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
5997   {
5998     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
5999     \dim_set:Nn \l_@@_y_initial_dim
6000     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6001   }
6002   \cs_if_exist:cT
6003   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6004   {
6005     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6006     \dim_set:Nn \l_@@_y_final_dim
6007     { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6008   }
6009 }
6010 \dim_set:Nn \l_tmpa_dim
6011 {
6012   \l_@@_y_initial_dim - \l_@@_y_final_dim +
6013   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6014 }
6015 \dim_set_eq:NN \nulldelimiterspace \c_zero_dim

```

We will draw the rules in the `\SubMatrix`.

```

6016 \group_begin:
6017 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6018 \tl_if_empty:NF \l_@@_rules_color_tl
6019 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6020 \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`.

```

6021 \seq_map_inline:Nn \g_@@_cols_vlism_seq
6022 {
6023   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6024   {
6025     \int_compare:nNnT
6026     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6027     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

6028       \@@_qpoint:n { col - ##1 }
6029       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6030       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6031       \pgfusepathqstroke
6032     }
6033   }
6034 }

```

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.

```

6035 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6036 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6037 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6038 {
6039   \bool_lazy_and:nnTF
6040   { \int_compare_p:nNn { ##1 } > 0 }
6041   {
6042     \int_compare_p:nNn
6043     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6044   {

```

```

6045         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6046         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6047         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6048         \pgfusepathqstroke
6049     }
6050     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
6051 }

```

Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

6052 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6053 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6054 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist {
6055     {
6056         \bool_lazy_and:nnTF
6057         { \int_compare_p:nNn { ##1 } > 0 }
6058         {
6059             \int_compare_p:nNn
6060             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6061         {
6062             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

6063     \group_begin:

```

We compute in `\l_tmpa_dim` the  $x$ -value of the left end of the rule.

```

6064         \dim_set:Nn \l_tmpa_dim
6065         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6066         \str_case:nn { #1 }
6067         {
6068             ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6069             [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
6070             \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6071         }
6072         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

6073         \dim_set:Nn \l_tmpb_dim
6074         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6075         \str_case:nn { #2 }
6076         {
6077             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6078             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
6079             \{ { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6080         }
6081         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6082         \pgfusepathqstroke
6083         \group_end:
6084     }
6085     { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
6086 }

```

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).

```

6087     \str_if_empty:NF \l_@@_submatrix_name_str
6088     {
6089         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
6090         \l_@@_x_initial_dim \l_@@_y_initial_dim
6091         \l_@@_x_final_dim \l_@@_y_final_dim
6092     }
6093     \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`.

```

6094   \begin { pgfscope }
6095   \pgftransformshift
6096   {
6097     \pgfpoint
6098     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6099     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6100   }
6101   \str_if_empty:NTF \l_@@_submatrix_name_str
6102   { \@@_node_left:nn #1 { } }
6103   { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
6104   \end { pgfscope }

```

Now, we deal with the right delimiter.

```

6105   \pgftransformshift
6106   {
6107     \pgfpoint
6108     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6109     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6110   }
6111   \str_if_empty:NTF \l_@@_submatrix_name_str
6112   { \@@_node_right:nn #2 { } }
6113   {
6114     \@@_node_right:nn #2 { \@@_env: - \l_@@_submatrix_name_str - right }
6115   }
6116   \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
6117   \flag_clear_new:n { nicematrix }
6118   \l_@@_code_tl
6119 }

```

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  $i-lj$  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`.

```

6120 \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.

```

6121 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
6122 {
6123   \use:e
6124   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
6125 }

```

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 -.

```

6126 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
6127 { #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.

```

6128 \tl_const:Nn \c_@@_integers_alist_tl

```

```

6129 {
6130   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
6131   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
6132   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
6133   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
6134 }

```

```

6135 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
6136 {

```

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 of a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

6137   \tl_if_empty:nTF { #2 }
6138   {
6139     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
6140     {
6141       \flag_raise:n { nicematrix }
6142       \int_if_even:nTF { \flag_height:n { nicematrix } }
6143       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
6144       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
6145     }
6146     { #1 }
6147   }

```

If there is an hyphen, we have to see whether we have a node of the form  $i-j$ ,  $\text{row-}i$  or  $\text{col-}j$ .

```

6148   { \@@_pgfpointanchor_iii:w { #1 } #2 }
6149 }

```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

6150 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
6151 {
6152   \str_case:nnF { #1 }
6153   {
6154     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
6155     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
6156   }

```

Now the case of a node of the form  $i-j$ .

```

6157   {
6158     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
6159     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
6160   }
6161 }

```

The command `\@@_node_left:nn` puts the left delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

6162 \cs_new_protected:Npn \@@_node_left:nn #1 #2
6163 {
6164   \pgfnode
6165   { rectangle }
6166   { east }
6167   {
6168     \nullfont
6169     \c_math_toggle_token
6170     \tl_if_empty:NF \l_@@_delimiters_color_tl
6171     { \color { \l_@@_delimiters_color_tl } }

```

```

6172     \left #1
6173     \vcenter
6174     {
6175         \nullfont
6176         \hrule \@height \l_tmpa_dim
6177             \@depth \c_zero_dim
6178             \@width \c_zero_dim
6179     }
6180     \right .
6181     \c_math_toggle_token
6182 }
6183 { #2 }
6184 { }
6185 }

```

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`).

```

6186 \cs_new_protected:Npn \@@_node_right:nn #1 #2
6187 {
6188     \pgfnode
6189     { rectangle }
6190     { west }
6191     {
6192         \nullfont
6193         \c_math_toggle_token
6194         \tl_if_empty:NF \l_@@_delimiters_color_tl
6195         { \color { \l_@@_delimiters_color_tl } }
6196         \left .
6197         \vcenter
6198         {
6199             \nullfont
6200             \hrule \@height \l_tmpa_dim
6201                 \@depth \c_zero_dim
6202                 \@width \c_zero_dim
6203         }
6204         \right #1
6205         \c_math_toggle_token
6206     }
6207     { #2 }
6208     { }
6209 }

```

## 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.

```

6210 \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.

```

6211 \bool_new:N \c_@@_footnote_bool

```

```

6212 \@@_msg_new:nnn { Unknown~option~for~package }
6213 {
6214   The~key~'\l_keys_key_str'~is~unknown. \\
6215   If~you~go~on,~it~will~be~ignored. \\
6216   For~a~list~of~the~available~keys,~type~H~<return>.
6217 }
6218 {
6219   The~available~keys~are~(in~alphabetic~order):~
6220   define-L-C-R,~
6221   footnote,~
6222   footnotehyper,~
6223   renew-dots,~and
6224   renew-matrix.
6225 }

```

Maybe we will completely delete the key 'transparent' in a future version.

```

6226 \@@_msg_new:nn { Key~transparent }
6227 {
6228   The~key~'transparent'~is~now~obsolete~(because~it's~name~
6229   is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
6230   and~'renew-matrix'.~However,~you~can~go~on.
6231 }
6232 \keys_define:nn { NiceMatrix / Package }
6233 {
6234   define-L-C-R .bool_set:N = \c_@@_define_L_C_R_bool ,
6235   define-L-C-R .default:n = true ,
6236   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
6237   renew-dots .value_forbidden:n = true ,
6238   renew-matrix .code:n = \@@_renew_matrix: ,
6239   renew-matrix .value_forbidden:n = true ,
6240   transparent .code:n =
6241   {
6242     \@@_renew_matrix:
6243     \bool_set_true:N \l_@@_renew_dots_bool
6244     \@@_error:n { Key~transparent }
6245   } ,
6246   transparent .value_forbidden:n = true,
6247   footnote .bool_set:N = \c_@@_footnote_bool ,
6248   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
6249   unknown .code:n = \@@_error:n { Unknown~option~for~package }
6250 }
6251 \ProcessKeysOptions { NiceMatrix / Package }

6252 \@@_msg_new:nn { footnote~with~footnotehyper~package }
6253 {
6254   You~can't~use~the~option~'footnote'~because~the~package~
6255   footnotehyper~has~already~been~loaded.~
6256   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
6257   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6258   of~the~package~footnotehyper.\\
6259   If~you~go~on,~the~package~footnote~won't~be~loaded.
6260 }
6261 \@@_msg_new:nn { footnotehyper~with~footnote~package }
6262 {
6263   You~can't~use~the~option~'footnotehyper'~because~the~package~
6264   footnote~has~already~been~loaded.~
6265   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
6266   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6267   of~the~package~footnote.\\
6268   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
6269 }

```

```

6270 \bool_if:NT \c_@@_footnote_bool
6271 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6272 \@ifclassloaded { beamer }
6273 { \bool_set_false:N \c_@@_footnote_bool }
6274 {
6275   \@ifpackageloaded { footnotehyper }
6276   { \@@_error:n { footnote-with-footnotehyper-package } }
6277   { \usepackage { footnote } }
6278 }
6279 }

6280 \bool_if:NT \c_@@_footnotehyper_bool
6281 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6282 \@ifclassloaded { beamer }
6283 { \bool_set_false:N \c_@@_footnote_bool }
6284 {
6285   \@ifpackageloaded { footnote }
6286   { \@@_error:n { footnotehyper-with-footnote-package } }
6287   { \usepackage { footnotehyper } }
6288 }
6289 \bool_set_true:N \c_@@_footnote_bool
6290 }

```

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`.

```

6291 \@@_msg_new:nn { key except-corners }
6292 {
6293   The~key~'except-corners'~has-been-deleted-for~the~command~\token_to_str:N
6294   \arraycolor\ in~the~\token_to_str:N \CodeBefore.~You~should~instead~use~
6295   the~key~'corners'~in~your~\@@_full_name_env:.\
6296   If~you~go~on,~this~key~will~be~ignored.
6297 }

6298 \seq_new:N \c_@@_types_of_matrix_seq
6299 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
6300 {
6301   NiceMatrix ,
6302   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
6303 }
6304 \seq_set_map_x:NNn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
6305 { \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`.

```

6306 \cs_new_protected:Npn \@@_error_too_much_cols:
6307 {
6308   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
6309   {
6310     \int_compare:nNnTF \l_@@_last_col_int = { -2 }

```

```

6311     { \@@_fatal:n { too-much-cols-for-matrix } }
6312     {
6313         \bool_if:NF \l_@@_last_col_without_value_bool
6314         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
6315     }
6316 }
6317 { \@@_fatal:n { too-much-cols-for-array } }
6318 }

```

The following command must *not* be protected since it's used in an error message.

```

6319 \cs_new:Npn \@@_message_hdotsfor:
6320 {
6321     \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
6322     { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is~incorrect.}
6323 }
6324 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
6325 {
6326     You-try-to-use-more-columns-than-allowed-by-your~
6327     \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
6328     columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
6329     exterior~columns).~This~error~is~fatal.
6330 }
6331 \@@_msg_new:nn { too-much-cols-for-matrix }
6332 {
6333     You-try-to-use-more-columns-than-allowed-by-your~
6334     \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
6335     number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
6336     'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
6337     This~error~is~fatal.
6338 }

```

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`.

```

6339 \@@_msg_new:nn { too-much-cols-for-array }
6340 {
6341     You-try-to-use-more-columns-than-allowed-by-your~
6342     \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
6343     \int_use:N \g_@@_static_num_of_col_int\
6344     ~(plus~the~potential~exterior~ones).~
6345     This~error~is~fatal.
6346 }
6347 \@@_msg_new:nn { last-col-not-used }
6348 {
6349     The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
6350     in~your~\@@_full_name_env:.\~However,~you~can~go~on.
6351 }
6352 \@@_msg_new:nn { columns-not-used }
6353 {
6354     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
6355     \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\
6356     You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
6357 }
6358 \@@_msg_new:nn { in-first-col }
6359 {
6360     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\
6361     If~you~go~on,~this~command~will~be~ignored.
6362 }
6363 \@@_msg_new:nn { in-last-col }
6364 {
6365     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\
6366     If~you~go~on,~this~command~will~be~ignored.
6367 }

```



```

6368 \@@_msg_new:nn { in-first-row }
6369 {
6370     You~can't~use~the~command~#1~in~the~first~row~(number~0)~of~the~array.\\
6371     If~you~go~on,~this~command~will~be~ignored.
6372 }
6373 \@@_msg_new:nn { in-last-row }
6374 {
6375     You~can't~use~the~command~#1~in~the~last~row~(exterior)~of~the~array.\\
6376     If~you~go~on,~this~command~will~be~ignored.
6377 }
6378 \@@_msg_new:nn { double-closing-delimiter }
6379 {
6380     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
6381     delimiter.~This~delimiter~will~be~ignored.
6382 }
6383 \@@_msg_new:nn { delimiter~after~opening }
6384 {
6385     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
6386     delimiter.~This~delimiter~will~be~ignored.
6387 }
6388 \@@_msg_new:nn { bad-option-for~line-style }
6389 {
6390     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
6391     is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
6392 }
6393 \@@_msg_new:nn { Unknown-key-for~xdots }
6394 {
6395     As~for~now,~there~is~only~three~key~available~here:~'color',~'line-style'~
6396     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
6397     this~key~will~be~ignored.
6398 }
6399 \@@_msg_new:nn { Unknown-key-for~rowcolors }
6400 {
6401     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
6402     (and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
6403     this~key~will~be~ignored.
6404 }
6405 \@@_msg_new:nn { ampersand~in~light-syntax }
6406 {
6407     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
6408     you~have~used~the~key~'light-syntax'.~This~error~is~fatal.
6409 }
6410 \@@_msg_new:nn { SubMatrix~too~large }
6411 {
6412     Your~command~\token_to_str:N \SubMatrix\
6413     can't~be~drawn~because~your~matrix~is~too~small.\\
6414     If~you~go~on,~this~command~will~be~ignored.
6415 }
6416 \@@_msg_new:nn { double-backslash~in~light-syntax }
6417 {
6418     You~can't~use~\token_to_str:N \\~to~separate~rows~because~you~have~used~
6419     the~key~'light-syntax'.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
6420     (set~by~the~key~'end-of-row').~This~error~is~fatal.
6421 }
6422 \@@_msg_new:nn { standard-cline~in~document }
6423 {
6424     The~key~'standard-cline'~is~available~only~in~the~preamble.\\
6425     If~you~go~on~this~command~will~be~ignored.
6426 }

```

```

6427 \@@_msg_new:nn { old~column~type }
6428 {
6429     The~column~type~'#1'~is~no~longer~defined~in~'nicematrix'.~
6430     Since~version~5.0,~you~have~to~use~'l',~'c'~and~'r'~instead~of~'L',~
6431     'C'~and~'R'.~You~can~also~use~the~key~'define-L-C-R'.\\
6432     This~error~is~fatal.
6433 }
6434 \@@_msg_new:nn { bad~value~for~baseline }
6435 {
6436     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
6437     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
6438     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
6439     If~you~go~on,~a~value~of~1~will~be~used.
6440 }
6441 \@@_msg_new:nn { Invalid~name~format }
6442 {
6443     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
6444     \SubMatrix.\\
6445     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
6446     If~you~go~on,~this~key~will~be~ignored.
6447 }
6448 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
6449 {
6450     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
6451     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
6452     number~is~not~valid.~If~you~go~on,~it~will~be~ignored.
6453 }
6454 \@@_msg_new:nn { impossible~delimiter }
6455 {
6456     It's~impossible~to~draw~the~#1~delimiter~of~your~
6457     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
6458     in~that~column.
6459     \bool_if:NT \l_@@_submatrix_slim_bool
6460     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
6461     If~you~go~on,~this~\token_to_str:N \SubMatrix\ will~be~ignored.
6462 }
6463 \@@_msg_new:nn { empty~environment }
6464 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }
6465 \@@_msg_new:nn { Delimiter~with~small }
6466 {
6467     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
6468     because~the~key~'small'~is~in~force.\\
6469     This~error~is~fatal.
6470 }
6471 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
6472 {
6473     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code~after'~
6474     can't~be~executed~because~a~cell~doesn't~exist.\\
6475     If~you~go~on~this~command~will~be~ignored.
6476 }
6477 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
6478 {
6479     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
6480     in~this~\@@_full_name_env:.\
6481     If~you~go~on,~this~key~will~be~ignored.\\
6482     For~a~list~of~the~names~already~used,~type~H~<return>.
6483 }
6484 {
6485     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
6486     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
6487 }

```

```

6488 \@@_msg_new:nn { r-or-l-with-preamble }
6489 {
6490     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:.~
6491     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
6492     your~\@@_full_name_env:.\
6493     If~you~go~on,~this~key~will~be~ignored.
6494 }
6495 \@@_msg_new:nn { Hdotsfor~in~col-0 }
6496 {
6497     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
6498     the~array.~This~error~is~fatal.
6499 }
6500 \@@_msg_new:nn { bad~corner }
6501 {
6502     #1~is~an~incorrect~specification~for~a~corner~(in~the~keys~
6503     'corners'~and~'except-corners').~The~available~
6504     values~are:~NW,~SW,~NE~and~SE.\
6505     If~you~go~on,~this~specification~of~corner~will~be~ignored.
6506 }
6507 \@@_msg_new:nn { bad~border }
6508 {
6509     #1~is~an~incorrect~specification~for~a~border~(in~the~key~
6510     'borders'~of~the~command~\token_to_str:N \Block).~The~available~
6511     values~are:~left,~right,~top~and~bottom.\
6512     If~you~go~on,~this~specification~of~border~will~be~ignored.
6513 }
6514 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
6515 {
6516     In~the~\@@_full_name_env:,~you~must~use~the~key~
6517     'last-col'~without~value.\
6518     However,~you~can~go~on~for~this~time~
6519     (the~value~'\l_keys_value_tl'~will~be~ignored).
6520 }
6521 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
6522 {
6523     In~\NiceMatrixoptions,~you~must~use~the~key~
6524     'last-col'~without~value.\
6525     However,~you~can~go~on~for~this~time~
6526     (the~value~'\l_keys_value_tl'~will~be~ignored).
6527 }
6528 \@@_msg_new:nn { Block-too-large-1 }
6529 {
6530     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
6531     too~small~for~that~block. \
6532 }
6533 \@@_msg_new:nn { Block-too-large-2 }
6534 {
6535     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
6536     \g_@@_static_num_of_col_int\
6537     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
6538     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
6539     (&)~at~the~end~of~the~first~row~of~your~
6540     \@@_full_name_env:.\
6541     If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
6542 }
6543 \@@_msg_new:nn { unknown~column~type }
6544 {
6545     The~column~type~'#1'~in~your~\@@_full_name_env:\
6546     is~unknown. \
6547     This~error~is~fatal.
6548 }

```

```

6549 \@@_msg_new:nn { tabularnote~forbidden }
6550 {
6551   You~can't~use~the~command~\token_to_str:N\tabularnote\
6552   ~in~a~\@@_full_name_env:~This~command~is~available~only~in~
6553   \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
6554   If~you~go~on,~this~command~will~be~ignored.
6555 }
6556 \@@_msg_new:nn { borders~forbidden }
6557 {
6558   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
6559   because~the~option~'rounded-corners'~
6560   is~in~force~with~a~non-zero~value.\\
6561   If~you~go~on,~this~key~will~be~ignored.
6562 }
6563 \@@_msg_new:nn { bottomrule~without~booktabs }
6564 {
6565   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
6566   loaded~'booktabs'.\\
6567   If~you~go~on,~this~key~will~be~ignored.
6568 }
6569 \@@_msg_new:nn { enumitem~not~loaded }
6570 {
6571   You~can't~use~the~command~\token_to_str:N\tabularnote\
6572   ~because~you~haven't~loaded~'enumitem'.\\
6573   If~you~go~on,~this~command~will~be~ignored.
6574 }
6575 \@@_msg_new:nn { Wrong~last~row }
6576 {
6577   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
6578   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
6579   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
6580   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
6581   without~value~(more~compilations~might~be~necessary).
6582 }
6583 \@@_msg_new:nn { Yet~in~env }
6584 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }
6585 \@@_msg_new:nn { Outside~math~mode }
6586 {
6587   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
6588   (and~not~in~\token_to_str:N \vcenter).\\
6589   This~error~is~fatal.
6590 }
6591 \@@_msg_new:nn { One~letter~allowed }
6592 {
6593   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
6594   If~you~go~on,~it~will~be~ignored.
6595 }
6596 \@@_msg_new:nnn { Unknown~key~for~Block }
6597 {
6598   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
6599   \Block.\\ If~you~go~on,~it~will~be~ignored. \\
6600   For~a~list~of~the~available~keys,~type~H<return>.
6601 }
6602 {
6603   The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
6604   hvlines,~l,~line-width,~rounded-corners,~r~and~t.
6605 }
6606 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
6607 {
6608   The~key~'\l_keys_key_str'~is~unknown.\\

```

```

6609     If~you~go~on,~it~will~be~ignored. \\
6610     For~a~list~of~the~available~keys~in~\token_to_str:N
6611     \CodeAfter,~type~H~<return>.
6612 }
6613 {
6614     The~available~keys~are~(in~alphabetic~order):~
6615     delimiters/color,~
6616     rules~(with~the~subkeys~'color'~and~'width'),~
6617     sub-matrix~(several~subkeys)~
6618     and~xdots~(several~subkeys).~
6619     The~latter~is~for~the~command~\token_to_str:N \line.
6620 }
6621 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
6622 {
6623     The~key~'\l_keys_key_str'~is~unknown.\\
6624     If~you~go~on,~this~key~will~be~ignored. \\
6625     For~a~list~of~the~available~keys~in~\token_to_str:N
6626     \SubMatrix,~type~H~<return>.
6627 }
6628 {
6629     The~available~keys~are~(in~alphabetic~order):~
6630     'delimiters/color',~
6631     'extra-height',~
6632     'hlines',~
6633     'hvlines',~
6634     'left-xshift',~
6635     'name',~
6636     'right-xshift',~
6637     'rules'~(with~the~subkeys~'color'~and~'width'),~
6638     'slim',~
6639     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
6640     and~'right-xshift').\\
6641 }
6642 \@@_msg_new:nnn { Unknown~key~for~notes }
6643 {
6644     The~key~'\l_keys_key_str'~is~unknown.\\
6645     If~you~go~on,~it~will~be~ignored. \\
6646     For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
6647 }
6648 {
6649     The~available~keys~are~(in~alphabetic~order):~
6650     bottomrule,~
6651     code-after,~
6652     code-before,~
6653     enumitem-keys,~
6654     enumitem-keys-para,~
6655     para,~
6656     label-in-list,~
6657     label-in-tabular~and~
6658     style.
6659 }
6660 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
6661 {
6662     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
6663     \token_to_str:N \NiceMatrixOptions. \\
6664     If~you~go~on,~it~will~be~ignored. \\
6665     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6666 }
6667 {
6668     The~available~keys~are~(in~alphabetic~order):~
6669     allow-duplicate-names,~
6670     cell-space-bottom-limit,~
6671     cell-space-limits,~

```

```

6672     cell-space-top-limit,~
6673     code-for-first-col,~
6674     code-for-first-row,~
6675     code-for-last-col,~
6676     code-for-last-row,~
6677     corners,~
6678     create-extra-nodes,~
6679     create-medium-nodes,~
6680     create-large-nodes,~
6681     delimiters~(several~subkeys),~
6682     end-of-row,~
6683     first-col,~
6684     first-row,~
6685     hlines,~
6686     hvlines,~
6687     last-col,~
6688     last-row,~
6689     left-margin,~
6690     letter-for-dotted-lines,~
6691     light-syntax,~
6692     notes~(several~subkeys),~
6693     nullify-dots,~
6694     renew-dots,~
6695     renew-matrix,~
6696     right-margin,~
6697     rules~(with~the~subkeys~'color'~and~'width'),~
6698     small,~
6699     sub-matrix~(several~subkeys),
6700     vlines,~
6701     xdots~(several~subkeys).
6702 }

6703 \@@_msg_new:nnn { Unknown~option~for~NiceArray }
6704 {
6705     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
6706     \{NiceArray\}. \\
6707     If~you~go~on,~it~will~be~ignored. \\
6708     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6709 }
6710 {
6711     The~available~keys~are~(in~alphabetic~order):~
6712     b,~
6713     baseline,~
6714     c,~
6715     cell-space-bottom-limit,~
6716     cell-space-limits,~
6717     cell-space-top-limit,~
6718     code-after,~
6719     code-for-first-col,~
6720     code-for-first-row,~
6721     code-for-last-col,~
6722     code-for-last-row,~
6723     colortbl-like,~
6724     columns-width,~
6725     corners,~
6726     create-extra-nodes,~
6727     create-medium-nodes,~
6728     create-large-nodes,~
6729     delimiters/color,~
6730     extra-left-margin,~
6731     extra-right-margin,~
6732     first-col,~
6733     first-row,~
6734     hlines,~

```

```

6735     hvlines,~
6736     last-col,~
6737     last-row,~
6738     left-margin,~
6739     light-syntax,~
6740     name,~
6741     notes/bottomrule,~
6742     notes/para,~
6743     nullify-dots,~
6744     renew-dots,~
6745     right-margin,~
6746     rules~(with~the~subkeys~'color'~and~'width'),~
6747     small,~
6748     t,~
6749     tabularnote,~
6750     vlines,~
6751     xdots/color,~
6752     xdots/shorten~and~
6753     xdots/line-style.
6754 }

```

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).

```

6755 \@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
6756 {
6757   The~key~'\l_keys_key_str'~is~unknown~for~the~
6758   \@@_full_name_env:. \\\
6759   If~you~go~on,~it~will~be~ignored. \\\
6760   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6761 }
6762 {
6763   The~available~keys~are~(in~alphabetic~order):~
6764   b,~
6765   baseline,~
6766   c,~
6767   cell-space-bottom-limit,~
6768   cell-space-limits,~
6769   cell-space-top-limit,~
6770   code-after,~
6771   code-for-first-col,~
6772   code-for-first-row,~
6773   code-for-last-col,~
6774   code-for-last-row,~
6775   colortbl-like,~
6776   columns-width,~
6777   corners,~
6778   create-extra-nodes,~
6779   create-medium-nodes,~
6780   create-large-nodes,~
6781   delimiters~(several~subkeys),~
6782   extra-left-margin,~
6783   extra-right-margin,~
6784   first-col,~
6785   first-row,~
6786   hlines,~
6787   hvlines,~
6788   l,~
6789   last-col,~
6790   last-row,~
6791   left-margin,~
6792   light-syntax,~
6793   name,~
6794   nullify-dots,~

```

```

6795     r,~
6796     renew-dots,~
6797     right-margin,~
6798     rules~(with~the~subkeys~'color'~and~'width'),~
6799     small,~
6800     t,~
6801     vlines,~
6802     xdots/color,~
6803     xdots/shorten~and~
6804     xdots/line-style.
6805 }

6806 \@@_msg_new:nnn { Unknown~option~for~NiceTabular }
6807 {
6808   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
6809   \{NiceTabular\}. \\
6810   If~you~go~on,~it~will~be~ignored. \\
6811   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6812 }
6813 {
6814   The~available~keys~are~(in~alphabetic~order):~
6815   b,~
6816   baseline,~
6817   c,~
6818   cell-space-bottom-limit,~
6819   cell-space-limits,~
6820   cell-space-top-limit,~
6821   code-after,~
6822   code-for-first-col,~
6823   code-for-first-row,~
6824   code-for-last-col,~
6825   code-for-last-row,~
6826   colortbl-like,~
6827   columns-width,~
6828   corners,~
6829   create-extra-nodes,~
6830   create-medium-nodes,~
6831   create-large-nodes,~
6832   extra-left-margin,~
6833   extra-right-margin,~
6834   first-col,~
6835   first-row,~
6836   hlines,~
6837   hvlines,~
6838   last-col,~
6839   last-row,~
6840   left-margin,~
6841   light-syntax,~
6842   name,~
6843   notes/bottomrule,~
6844   notes/para,~
6845   nullify-dots,~
6846   renew-dots,~
6847   right-margin,~
6848   rules~(with~the~subkeys~'color'~and~'width'),~
6849   t,~
6850   tabularnote,~
6851   vlines,~
6852   xdots/color,~
6853   xdots/shorten~and~
6854   xdots/line-style.
6855 }

6856 \@@_msg_new:nnn { Duplicate~name }
6857 {

```



```

6858 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
6859 the~same~environment~name~twice.~You~can~go~on,~but,~
6860 maybe,~you~will~have~incorrect~results~especially~
6861 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
6862 message~again,~use~the~key~'allow-duplicate-names'~in~
6863 '\token_to_str:N \NiceMatrixOptions'.\\
6864 For~a~list~of~the~names~already~used,~type~H~<return>. \\
6865 }
6866 {
6867 The~names~already~defined~in~this~document~are:~
6868 \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
6869 }
6870 \@@_msg_new:nn { Option~auto~for~columns~width }
6871 {
6872 You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
6873 If~you~go~on,~the~key~will~be~ignored.
6874 }

```

## 18 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 StackExchange<sup>62</sup>, Tikz externalization is now deactivated in the environments of the package `nicematrix`.<sup>63</sup>

## 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](http://www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end)

## Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

## 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`.

---

<sup>62</sup>cf. [tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package](https://tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package)

<sup>63</sup>Before 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 2.2.1 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 StackExchange<sup>64</sup>, 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.

---

<sup>64</sup>cf. [tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize](https://tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize)

## 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`.

## 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).

# 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_i</code> ..... 5005, 5006, 5010
<code>\@@_Block:</code> .....	1211, 5000	<code>\@@_Block_ii:nnnnn</code> ..... 5010, 5011

\@@_Block_iv:nnnnn	5041, 5045	\@@_adjust_pos_of_blocks_seq_i:nnnn	2638, 2640
\@@_Block_iv:nnnnnn	5225, 5227	\@@_adjust_size_box:	937, 963, 1802, 2351, 2396
\@@_Block_v:nnnnn	5042, 5146	\@@_adjust_to_submatrix:nn	2920, 3023, 3062, 3143, 3218, 3270
\@@_Block_v:nnnnnn	5254, 5257	\@@_adjust_to_submatrix:nnnnnn	2927, 2929
\@@_Cdots	1136, 1201, 3519	\@@_after_array:	1592, 2482
\g_@@_Cdots_lines_tl	1228, 2672	\g_@@_after_col_zero_bool	298, 1103, 2328, 3660
\@@_Cell:	186, 205, 859, 1724, 1771, 1793, 2449, 3619, 3620, 3621, 3622, 3623, 3624, 3625, 3626, 3627, 3628, 3629, 3630	\@@_analyze_end:Nn	2125, 2170
\@@_CodeAfter:	1215, 5778	\l_@@_argspec_tl	3501, 3502, 3503, 3519, 3535, 3551, 3575, 3674, 3675, 3676, 3750, 3751, 3752, 3828, 3829, 3830
\@@_CodeAfter_i:n	861, 2327, 2372, 5778, 5779, 5789	\@@_array:	1057, 2126, 2153
\@@_CodeAfter_ii:n	5782, 5784	\@@_arraycolor	1366, 3990
\@@_CodeAfter_keys:	2609, 2633	\c_@@_arydshln_loaded_bool	24, 31, 1710
\@@_CodeBefore:w	1403, 1405	\l_@@_auto_columns_width_bool	495, 635, 2237, 2241, 4761
\@@_CodeBefore_keys:	1383, 1400	\l_@@_baseline_tl	486, 487, 628, 629, 630, 631, 1070, 1525, 1938, 1950, 1955, 1957, 1962, 1967, 2049, 2050, 2054, 2059, 2061, 2066
\@@_Ddots	1138, 1203, 3551	\@@_begin_of_NiceMatrix:nn	2438, 2459
\g_@@_Ddots_lines_tl	1231, 2670	\@@_begin_of_row:	864, 887, 2329
\g_@@_HVdotsfor_lines_tl	1233, 2668, 3678, 3754, 6321	\l_@@_block_auto_columns_width_bool	1469, 2242, 4754, 4759, 4769, 4779
\@@_Hdotsfor:	1141, 1208, 3654	\g_@@_block_box_int	336, 1449, 5047, 5061, 5063, 5099, 5111, 5123, 5132, 5142
\@@_Hdotsfor:nnnn	3680, 3692	\g_@@_blocks_dp_dim	266, 945, 948, 949, 5126, 5129
\@@_Hdotsfor_i	3663, 3669, 3676	\g_@@_blocks_ht_dim	265, 951, 954, 955, 5117, 5120
\@@_Hline:	1206, 4497	\g_@@_blocks_seq	312, 1471, 1988, 5136, 5148, 5225
\@@_Hline_i:n	4497, 4498, 4504	\g_@@_blocks_wd_dim	264, 939, 942, 943, 5105, 5108
\@@_Hline_ii:nn	4501, 4504	\c_@@_booktabs_loaded_bool	25, 34, 1151, 2020
\@@_Hline_iii:n	4502, 4505	\l_@@_borders_clist	326, 5204, 5284, 5572, 5588, 5590, 5592, 5594, 5613, 5625
\@@_Hspace:	1207, 3605	\@@_cartesian_path:	3923, 3938, 4062, 4074, 4167
\@@_Iddots	1139, 1204, 3575	\@@_cartesian_path:n	3968, 4109, 4167
\g_@@_Iddots_lines_tl	1232, 2671	\l_@@_cell_box	865, 911, 913, 919, 925, 928, 932, 941, 942, 947, 948, 953, 954, 964, 965, 966, 967, 969, 972, 976, 978, 997, 1012, 1014, 1021, 1022, 1035, 1153, 1245, 1247, 1792, 1803, 2330, 2354, 2357, 2359, 2376, 2399, 2403, 5319, 5423, 5457, 5711
\@@_Ldots	1135, 1140, 1200, 3503	\l_@@_cell_space_bottom_limit_dim	475, 547, 967
\g_@@_Ldots_lines_tl	1229, 2673	\l_@@_cell_space_top_limit_dim	474, 545, 965
\l_@@_Matrix_bool	272, 1583, 1599, 2440	\l_@@_cell_type_tl	262, 263, 1724, 1794, 5025, 5027
\l_@@_NiceArray_bool	269, 392, 1293, 1481, 1523, 1636, 1642, 1654, 2416, 4366, 4367, 4489, 4490, 4707, 4714, 5696	\@@_cellcolor	1361, 3970, 3982, 3983
\g_@@_NiceMatrixBlock_int	258, 4766, 4771, 4774, 4785	\@@_cellcolor_tabular	1145, 4192
\l_@@_NiceTabular_bool	156, 270, 866, 1062, 1382, 1385, 1456, 1556, 1560, 1643, 1655, 2469, 2478, 2608, 2611, 5073, 5155, 5704	\g_@@_cells_seq	2164, 2165, 2166, 2168
\@@_NotEmpty:	1217, 2463	\@@_chessboardcolors	1368, 3975
\@@_OnlyMainNiceMatrix:n	1213, 4227	\@@_cline	135, 1199
\@@_OnlyMainNiceMatrix_i:n	4230, 4237, 4240	\@@_cline_i:nn	136, 137, 149, 152
\@@_SubMatrix	2590, 5924	\@@_cline_i:w	137, 138
\@@_SubMatrix_in_code_before	1369, 5908	\l_@@_code_before_bool	302, 625, 652, 1077, 1239, 1315, 1477, 2184, 2201, 2219, 2250, 2276, 2303, 2528, 2626, 2628
\@@_Vdots	1137, 1202, 3535		
\g_@@_Vdots_lines_tl	1230, 2669		
\@@_Vdotsfor:	1209, 3752		
\@@_Vdotsfor:nnnn	3756, 3767		
\@@_W:	1602, 1682		
\@@_actually_color:	1384, 3892		
\@@_actually_diagbox:nnnnnn	5052, 5311, 5716, 5732		
\@@_actually_draw_Cdots:	3075, 3079		
\@@_actually_draw_Ddots:	3225, 3229		
\@@_actually_draw_Iddots:	3277, 3281		
\@@_actually_draw_Ldots:	3036, 3040, 3743		
\@@_actually_draw_Vdots:	3157, 3161, 3818		
\@@_adapt_S_column:	215, 230, 1455		
\@@_add_to_colors_seq:nn	3879, 3891, 3914, 3929, 3944, 3953		
\@@_adjust_pos_of_blocks_seq:	2577, 2635		



\l_@@_code_before_tl .....	301, 624, 1314, 1383, 1478	\@@_define_com:nnn .....	5680, 5688, 5689, 5690, 5691, 5692
\l_@@_code_for_first_col_tl .....	564, 2341	\@@_delimiter:nnn .....	1829, 1840, 1862, 1870, 1883, 1889, 1895, 5792
\l_@@_code_for_first_row_tl .....	568, 875, 5395	\l_@@_delimiters_color_tl .....	507, 711, 1396, 1548, 1549, 1566, 1567, 5772, 5830, 5831, 5858, 6170, 6171, 6194, 6195
\l_@@_code_for_last_col_tl .....	566, 2385	\l_@@_delimiters_max_width_bool .....	508, 709, 1571
\l_@@_code_for_last_row_tl .....	570, 882, 5398	\g_@@_delta_x_one_dim .....	2558, 3252, 3262
\l_@@_code_tl .....	293, 5903, 6118	\g_@@_delta_x_two_dim .....	2560, 3300, 3310
\l_@@_col_max_int .....	321, 2787, 2798, 2866, 2925, 2942	\g_@@_delta_y_one_dim .....	2559, 3254, 3262
\l_@@_col_min_int .....	320, 2792, 2855, 2860, 2923, 2940	\g_@@_delta_y_two_dim .....	2561, 3302, 3310
\g_@@_col_total_int .....	261, 980, 1224, 1509, 2269, 2270, 2306, 2310, 2315, 2316, 2375, 2486, 2489, 2494, 2501, 2545, 2697, 3110, 3128, 3188, 3650, 3651, 3813, 4217, 4812, 4822, 4856, 4943, 5237, 5430, 5944, 5993	\@@_diagbox:nn .....	1216, 5712
\l_@@_color_tl .....	328, 4997, 5065, 5067	\@@_dotfill: .....	5701
\g_@@_colors_seq .....	1379, 3882, 3886, 3887, 3896	\@@_dotfill_i: .....	5706, 5708
\@@_colortbl_like: .....	1143, 1218	\@@_dotfill_ii: .....	5705, 5708, 5709
\l_@@_colortbl_like_bool .....	472, 651, 1218, 1625	\@@_dotfill_iii: .....	5709, 5710
\c_@@_colortbl_loaded_bool .....	88, 92, 1168	\@@_double_int_eval:n .....	3824, 3838, 3839
\l_@@_cols_tl .....	3922, 3937, 3967, 4001, 4009, 4010, 4115, 4118	\g_@@_dp_ante_last_row_dim .....	890, 1184
\g_@@_cols_vlism_seq .....	281, 1620, 1701, 6021	\g_@@_dp_last_row_dim .....	890, 891, 1187, 1188, 1246, 1247, 1542
\@@_columncolor .....	1367, 3925	\g_@@_dp_row_zero_dim .....	910, 911, 1178, 1179, 1535, 2043, 2082
\@@_columncolor:n .....	3931, 3934	\@@_draw_Cdots:nnn .....	3060
\@@_columncolor_preamble .....	1147, 4215	\@@_draw_Ddots:nnn .....	3216
\c_@@_columncolor_regex .....	233, 1628	\@@_draw_Iddots:nnn .....	3268
\l_@@_columns_width_dim .....	259, 636, 778, 2238, 2244, 4767, 4773	\@@_draw_Ldots:nnn .....	3021
\g_@@_com_or_env_str .....	285, 288	\@@_draw_Vdots:nnn .....	3141
\@@_computations_for_large_nodes: .....	4883, 4896, 4901	\@@_draw_blocks: .....	1988, 5222
\@@_computations_for_medium_nodes: .....	4803, 4872, 4882, 4893	\@@_draw_dotted_lines: .....	2575, 2657
\@@_compute_a_corner:nnnnnn .....	4568, 4570, 4572, 4574, 4590	\@@_draw_dotted_lines_i: .....	2660, 2664
\@@_compute_corners: .....	2576, 4560	\l_@@_draw_first_bool .....	334, 3566, 3590, 3601
\@@_construct_preamble: .....	1290, 1596	\@@_draw_hlines: .....	2588, 4486
\l_@@_corners_cells_seq .....	316, 1373, 2582, 4112, 4152, 4301, 4307, 4314, 4426, 4432, 4439, 4562, 4578, 4585, 4646	\@@_draw_line: .....	3058, 3103, 3214, 3266, 3314, 3316, 3877, 4722, 4752
\l_@@_corners_clist .....	491, 613, 618, 4267, 4393, 4563	\@@_draw_line_ii:nn .....	3857, 3861
\@@_create_col_nodes: .....	2129, 2157, 2176	\@@_draw_line_iii:nn .....	3864, 3868
\@@_create_diag_nodes: .....	1350, 2553, 2739	\@@_draw_non_standard_dotted_line: .....	3322, 3324
\@@_create_extra_nodes: .....	1987, 2731, 4793	\@@_draw_non_standard_dotted_line:n .....	3327, 3330
\@@_create_large_nodes: .....	4801, 4877	\@@_draw_non_standard_dotted_line:nnn .....	3332, 3337, 3351
\@@_create_medium_and_large_nodes: .....	4798, 4888	\@@_draw_standard_dotted_line: .....	3321, 3352
\@@_create_medium_nodes: .....	4799, 4867	\@@_draw_standard_dotted_line_i: .....	3415, 3419
\@@_create_nodes: .....	4874, 4885, 4895, 4898, 4939	\l_@@_draw_tl .....	325, 5198, 5202, 5262, 5472, 5478, 5480, 5482, 5519, 5520
\@@_create_row_node: .....	1073, 1106, 1152	\@@_draw_vlines: .....	2589, 4363
\@@_cut_on_hyphen:w .....	3905, 3960, 3965, 4030, 4122, 4123, 4145, 4146, 4176, 4177, 5490, 5499, 5530, 5533, 5577, 5580, 5912, 5917, 5936, 5939	\g_@@_empty_cell_bool .....	309, 971, 981, 2364, 2411, 3517, 3533, 3549, 3573, 3596, 3607
\g_@@_ddots_int .....	2556, 3249, 3250	\@@_end_Cell: .....	191, 207, 958, 1726, 1781, 1798, 2449, 3619, 3620, 3621, 3622, 3623, 3624, 3625, 3626, 3627, 3628, 3629, 3630
\@@_def_env:nnn .....	2422, 2433, 2434, 2435, 2436, 2437	\l_@@_end_of_row_tl .....	504, 505, 558, 2149, 2150, 6419
\@@_define_L_C_R: .....	246, 1289	\c_@@_endpgfortikzpicture_tl .....	43, 47, 2661, 3865, 4693
\c_@@_define_L_C_R_bool .....	245, 1289, 6234	\c_@@_enumitem_loaded_bool .....	26, 37, 365, 679, 684, 695, 700
		\@@_env: .....	253, 257, 896, 902, 998, 1004, 1018, 1026, 1082, 1088, 1094, 1330, 1331, 1338, 1339, 1346, 1347,

1358, 1424, 1427, 1430, 1433, 2185, 2188,	833, 864, 1518, 1637, 2179, 2199, 2539,
2190, 2206, 2212, 2215, 2224, 2230, 2233,	2697, 3110, 3128, 3658, 4136, 4229, 4812,
2255, 2261, 2264, 2281, 2287, 2293, 2306,	4822, 4856, 4904, 4943, 5662, 5668, 5674, 5993
2310, 2316, 2599, 2699, 2703, 2709, 2716,	\l_@@_first_i_tl 5932, 5937, 5956, 5987,
2722, 2756, 2763, 2765, 2766, 2827, 2895,	5996, 5998, 6053, 6060, 6062, 6143, 6154, 6158
2959, 2970, 2983, 2986, 3005, 3008, 3113,	\l_@@_first_j_tl ..... 5933, 5938, 5960,
3116, 3131, 3134, 3706, 3724, 3781, 3799,	5962, 6023, 6036, 6043, 6045, 6144, 6155, 6159
3850, 3852, 3871, 3874, 4601, 4620, 4638,	\l_@@_first_row_int ..... 344, 345, 561,
4825, 4827, 4835, 4946, 4955, 4973, 5334,	837, 1222, 1533, 1969, 2040, 2068, 2079,
5341, 5345, 5359, 5364, 5376, 5381, 5382,	2537, 2695, 2980, 3002, 4805, 4819, 4846,
5385, 5402, 5436, 5807, 5810, 5960, 5962,	4903, 4941, 5338, 5356, 5660, 5804, 5957, 6437
5967, 5969, 5996, 5998, 6003, 6005, 6103, 6114	\c_@@_footnote_bool ..... 1444, 1594, 6211, 6247, 6270, 6273, 6283, 6289
\g_@@_env_int .. 252, 253, 255, 1255, 1258,	\c_@@_footnotehyper_bool . 6210, 6248, 6280
1273, 1276, 1322, 1325, 1335, 1336, 1343,	\@@_full_name_env: ..... 286, 6295, 6327, 6334, 6342, 6350, 6354,
1344, 1371, 1374, 1407, 1416, 1417, 1420,	6451, 6464, 6467, 6480, 6485, 6490, 6492,
1421, 1468, 1475, 1479, 2493, 2514, 2532,	6516, 6535, 6540, 6545, 6552, 6578, 6587, 6758
2535, 2548, 2622, 4017, 4020, 4045, 4584, 4982	\@@_hdottedline: ..... 1205, 4675
\@@_error:n ..... 12, 368, 393, 517, 527,	\@@_hdottedline:n ..... 4683, 4687
580, 705, 761, 771, 777, 786, 794, 812, 819,	\@@_hdottedline_i: ..... 4678, 4680
827, 828, 829, 835, 840, 841, 842, 853, 855,	\@@_hdottedline_i:n ..... 4692, 4696
856, 857, 1398, 1504, 1514, 1586, 1972,	\@@_hline:nn ..... 4374, 4494, 4513
2025, 2071, 3989, 4004, 5220, 5570, 5776,	\@@_hline_i:nn ..... 2586, 4377, 4380
5888, 5899, 5906, 5945, 6244, 6249, 6276, 6286	\@@_hline_i_complete:nn ..... 2586, 4484
\@@_error:nn ..... 13, 643, 1690, 1691, 1692, 1832,	\@@_hline_ii:nnnn ... 4402, 4413, 4446, 4485
1843, 1890, 1896, 3506, 3509, 3522, 3525,	\l_@@_hlines_clist ..... 340, 572,
3538, 3541, 3555, 3556, 3561, 3562, 3579,	586, 615, 1107, 1109, 1113, 2588, 4492, 4493
3580, 3585, 3586, 4576, 5575, 5893, 5975, 5978	\l_@@_hpos_of_block_tl .. 330, 331, 4987,
\@@_error:nnn ..... 14, 3855, 6050, 6085	4989, 4991, 5026, 5027, 5029, 5072, 5078,
\@@_error_too_much_cols: ..... 1664, 6306	5088, 5139, 5162, 5166, 5178, 5183, 5210,
\@@_everycr: ..... 1099, 1173, 1176	5212, 5214, 5404, 5416, 5427, 5431, 5438, 5450
\@@_everycr_i: ..... 1099, 1100	\g_@@_ht_last_row_dim ..... 892, 1185, 1186, 1244, 1245, 1541
\@@_exec_code_before: ..... 1239, 1377	\g_@@_ht_row_one_dim .. 918, 919, 1182, 1183
\@@_expand_clist:NN ..... 4115, 4116, 4168	\g_@@_ht_row_zero_dim ..... 912, 913, 1180, 1181, 1536, 2042, 2081
\l_@@_exterior_arraycolsep_bool ..... 488, 774, 1645, 1657	\@@_hvlines_block:nnn ..... 5278, 5526
\l_@@_extra_left_margin_dim ..... 502, 602, 1306, 2362	\l_@@_hvlines_block_bool .. 335, 5206, 5274
\l_@@_extra_right_margin_dim ..... 503, 603, 1496, 2407, 3191	\@@_i: ..... 4805, 4807,
\@@_extract_brackets ..... 5296, 5462	4808, 4809, 4810, 4819, 4825, 4827, 4828,
\@@_extract_coords_values: .... 4964, 4971	4829, 4830, 4835, 4836, 4837, 4838, 4846,
\@@_fatal:n .... 15, 277, 1459, 1817, 1851,	4849, 4851, 4852, 4853, 4905, 4907, 4910,
2134, 2138, 2140, 2173, 3665, 6311, 6314, 6317	4911, 4915, 4916, 4941, 4946, 4948, 4950,
\@@_fatal:nn ..... 16, 1715	4954, 4955, 4966, 4973, 4975, 4977, 4981, 4982
\l_@@_fill_tl ..... 324, 5196, 5294, 5296	\g_@@_iddots_int ..... 2557, 3297, 3298
\l_@@_final_i_int ..... 2565, 2774, 2779, 2782, 2807,	\l_@@_in_env_bool .... 268, 392, 1459, 1460
2815, 2819, 2828, 2836, 2916, 2971, 3052,	\c_@@_in_preamble_bool . 21, 22, 23, 675, 691
3125, 3131, 3134, 3697, 3725, 3793, 3803, 3805	\l_@@_initial_i_int ..... 2563,
\l_@@_final_j_int ..... 2566, 2775, 2780, 2787, 2792, 2798, 2808,	2772, 2847, 2850, 2875, 2883, 2887, 2896,
2816, 2820, 2829, 2837, 2917, 2972, 3005,	2904, 2914, 2960, 3045, 3091, 3093, 3107,
3008, 3016, 3242, 3718, 3728, 3730, 3772, 3801	3113, 3116, 3696, 3697, 3707, 3775, 3785, 3787
\l_@@_final_open_bool ..... 2568, 2781,	\l_@@_initial_j_int ..... 2564, 2773, 2848, 2855,
2785, 2788, 2795, 2801, 2805, 2821, 3049,	2860, 2866, 2876, 2884, 2888, 2897, 2905,
3084, 3089, 3100, 3164, 3174, 3179, 3200,	2915, 2961, 2983, 2986, 2994, 3181, 3183,
3239, 3289, 3423, 3438, 3469, 3470, 3695,	3188, 3234, 3700, 3710, 3712, 3771, 3772, 3783
3719, 3731, 3770, 3794, 3806, 3847, 4690, 4727	\l_@@_initial_open_bool ..... 2567, 2849, 2853, 2856, 2863, 2869,
\@@_find_extremities_of_line:nnnn ... 2769, 3026, 3065, 3146, 3221, 3273	2873, 2889, 3042, 3081, 3088, 3098, 3164,
\l_@@_first_col_int ..... 123, 136, 346, 347, 560,	3171, 3177, 3231, 3283, 3421, 3468, 3694,
	3701, 3713, 3769, 3776, 3788, 3846, 4689, 4726
	\@@_insert_tabularnotes: ..... 1992, 1995

\@@_instruction_of_type:nnn .....	\@@_line_i:nn .....
..... 1038, 3511, 3527, 3543, 3566, 3590	..... 3837, 3844
\c_@@_integers_alist_tl .....	\l_@@_line_width_dim .....
..... 6128, 6139	..... 329, 5208, 5473, 5511, 5522, 5528, 5540,
\l_@@_inter_dots_dim .....	..... 5567, 5587, 5602, 5604, 5614, 5615, 5617, 5628
..... 476, 477, 2572, 3426, 3433, 3444, 3452,	\@@_line_with_light_syntax:n ... 2156, 2160
..... 3459, 3464, 3476, 3484, 4718, 4720, 4748, 4750	\@@_line_with_light_syntax_i:n .....
\g_@@_internal_code_after_tl .....	..... 2155, 2161, 2162
..... 294, 1760, 1828,	\@@_math_toggle_token: .....
..... 1839, 1861, 1869, 1882, 1888, 1894, 1904,	..... 155, 960, 2331, 2348, 2377, 2393, 5756, 5760
..... 2604, 2605, 4512, 4682, 5050, 5309, 5714, 5920	\g_@@_max_cell_width_dim .....
\@@_intersect_our_row:nnnn .....	..... 968, 969, 1470, 2243, 4760, 4786
\@@_intersect_our_row_p:nnnn .....	\c_@@_max_l_dim .....
..... 4098	..... 3413, 3418
\@@_j: .....	\l_@@_medium_nodes_bool 498, 592, 4795, 5378
..... 4812, 4814,	\@@_message_hdotsfor: 6319, 6327, 6334, 6342
..... 4815, 4816, 4817, 4822, 4825, 4827, 4830,	\@@_msg_new:nn .....
..... 4832, 4833, 4835, 4838, 4840, 4841, 4856,	..... 17, 6226, 6252, 6261, 6291, 6324,
..... 4859, 4861, 4862, 4863, 4918, 4920, 4923,	..... 6331, 6339, 6347, 6352, 6358, 6363, 6368,
..... 4925, 4929, 4930, 4943, 4946, 4947, 4949,	..... 6373, 6378, 6383, 6388, 6393, 6399, 6405,
..... 4954, 4955, 4967, 4973, 4974, 4976, 4981, 4982	..... 6410, 6416, 6422, 6427, 6434, 6441, 6448,
\l_@@_l_dim .....	..... 6454, 6463, 6465, 6471, 6488, 6495, 6500,
..... 3399, 3400, 3413, 3414, 3426, 3432,	..... 6507, 6514, 6521, 6528, 6533, 6543, 6549,
..... 3443, 3451, 3459, 3464, 3476, 3477, 3484, 3485	..... 6556, 6563, 6569, 6575, 6583, 6585, 6591, 6870
\l_@@_large_nodes_bool 499, 593, 4797, 4801	\@@_msg_new:nnn ... 18, 6212, 6477, 6596,
\g_@@_last_col_found_bool .. 354, 1227,	..... 6606, 6621, 6642, 6660, 6703, 6755, 6806, 6856
..... 1510, 1578, 2268, 2297, 2373, 2485, 2542, 5428	\@@_msg_redirect_name:nn .....
\l_@@_last_col_int .....	..... 19, 780, 1590, 5243, 5246
..... 352,	\@@_multicolumn:nnn .....
..... 353, 762, 805, 807, 820, 836, 854, 1269,	..... 1210, 3611
..... 1275, 1282, 1328, 1513, 1649, 2445, 2447,	\g_@@_multicolumn_cells_seq .....
..... 2486, 2489, 2541, 3151, 3186, 3508, 3524,	..... 1220, 3638, 4830, 4838, 4960, 5343, 5361
..... 3562, 3586, 5230, 5235, 5236, 5237, 5240,	\g_@@_multicolumn_sizes_seq 1221, 3640, 4961
..... 5269, 5281, 5291, 5303, 5315, 5330, 5359,	\g_@@_name_env_str .....
..... 5364, 5372, 5387, 5664, 5670, 5676, 6310, 6328	..... 284,
\l_@@_last_col_without_value_bool ...	..... 289, 290, 1453, 1454, 2172, 2417, 2418,
..... 351, 804, 2487, 6313	..... 2426, 2427, 2456, 2467, 2475, 2629, 5684, 6308
\l_@@_last_empty_column_int .....	\l_@@_name_str . 497, 645, 898, 901, 1000,
..... 4611, 4612, 4625, 4631, 4644	..... 1003, 1090, 1093, 1253, 1262, 1265, 1271,
\l_@@_last_empty_row_int .....	..... 1280, 1283, 2189, 2190, 2214, 2215, 2232,
..... 4593, 4594, 4607, 4628	..... 2233, 2263, 2264, 2289, 2292, 2312, 2315,
\l_@@_last_i_tl .....	..... 2496, 2500, 2517, 2521, 4951, 4954, 4978, 4981
..... 5934,	\g_@@_names_seq .....
..... 5940, 5943, 5956, 5990, 6003, 6005, 6053, 6060	..... 267, 642, 644, 6868
\l_@@_last_j_tl .....	\l_@@_nb_cols_int .....
..... 5935, 5941, 5944, 5967, 5969, 6026, 6036, 6043	..... 5651, 5656, 5659, 5663, 5669, 5675
\l_@@_last_row_int .....	\l_@@_nb_rows_int .....
..... 348, 349, 562, 880, 926, 1119, 1240, 1250,	..... 5650, 5655, 5666
..... 1257, 1264, 1326, 1498, 1502, 1505, 1517,	\@@_newcolumn_type .....
..... 1539, 2151, 2152, 2337, 2338, 2382, 2383,	..... 1126, 1601, 1602
..... 2508, 3031, 3070, 3540, 3556, 3580, 4235,	\@@_node_for_cell: 977, 984, 1388, 2358, 2408
..... 4243, 5229, 5232, 5233, 5252, 5269, 5281,	\@@_node_for_multicolumn:nn .... 4962, 4969
..... 5291, 5303, 5314, 5328, 5386, 5397, 5672, 6577	\@@_node_left:nn .....
\l_@@_last_row_without_value_bool ...	..... 6102, 6103, 6162
..... 350, 1252, 1500, 2506	\@@_node_position: .....
\l_@@_left_delim_dim .....	..... 1338, 1340, 1346, 1348, 1428, 1434
..... 1291, 1295, 1300, 2117, 2360	\@@_node_position_i: .....
\g_@@_left_delim_tl .....	..... 1431, 1435
..... 1299, 1446, 1550, 1574, 1634, 1820, 1822, 4717	\@@_node_right:nn .....
\l_@@_left_margin_dim .....	..... 6112, 6114, 6186
..... 500, 596, 1305, 2361, 4708, 4934	\g_@@_not_empty_cell_bool 300, 975, 982, 2464
\l_@@_letter_for_dotted_lines_str ...	\@@_not_in_exterior:nnnn .....
..... 785, 796, 797, 1696	..... 4090
\l_@@_letter_vlism_tl .....	\@@_not_in_exterior_p:nnnn .....
..... 280, 579, 1699	..... 4022
\l_@@_light_syntax_bool .....	\l_@@_notes_above_space_dim .... 492, 493
..... 485, 556, 1308, 1491, 2509	\l_@@_notes_bottomrule_bool .....
\@@_light_syntax_i .....	..... 663, 823, 848, 2018
..... 2142, 2145	\l_@@_notes_code_after_tl .....
\@@_line .....	..... 661, 2027
..... 2603, 3830	\l_@@_notes_code_before_tl .....
	..... 659, 1999
	\@@_notes_label_in_list:n 361, 380, 388, 671
	\@@_notes_label_in_tabular:n . 360, 401, 668
	\l_@@_notes_para_bool .. 657, 821, 846, 2003
	\@@_notes_style:n .....
	..... 359, 362, 380, 388, 404, 409, 665

<code>\l_@@_nullify_dots_bool</code> .....	<code>\g_@@_pos_of_stroken_blocks_seq</code> .....
..... 494, 591, 3515, 3531, 3547, 3571, 3594	..... 315, 1473, 4265, 4391, 5271
<code>\l_@@_number_of_notes_int</code> 358, 395, 405, 415	<code>\g_@@_pos_of_xdots_seq</code> .....
<code>\@@_old_CT@arc@</code> ..... 1461, 2631	..... 314, 1474, 2581, 2912, 4263, 4389
<code>\@@_old_cdots</code> ..... 1193, 3532	<code>\@@_pre_array:</code> ..... 1236, 1316, 1488
<code>\@@_old_ddots</code> ..... 1195, 3572	<code>\@@_pre_array_i:w</code> ..... 1312, 1488
<code>\@@_old_dotfill</code> ..... 5700, 5703, 5711	<code>\@@_pre_array_ii:</code> ..... 1149, 1287
<code>\@@_old_dotfill:</code> ..... 1214	<code>\@@_pre_code_before:</code> ..... 1318, 1409
<code>\l_@@_old_iRow_int</code> ..... 295, 1155, 2677	<code>\c_@@_preamble_first_col_tl</code> .... 1638, 2323
<code>\@@_old_ialign:</code> ..... 1072, 1189, 5224	<code>\c_@@_preamble_last_col_tl</code> .... 1650, 2368
<code>\@@_old_iddots</code> ..... 1196, 3595	<code>\g_@@_preamble_tl</code> ..... 1448,
<code>\l_@@_old_jCol_int</code> ..... 296, 1158, 2678	1603, 1607, 1610, 1616, 1630, 1638, 1647,
<code>\@@_old_ldots</code> ..... 1192, 3516	1650, 1659, 1663, 1703, 1712, 1722, 1733,
<code>\@@_old_multicolumn</code> ..... 3610, 3614	1746, 1768, 1789, 1811, 1827, 1860, 1868,
<code>\@@_old_pgfpintanchor</code> .... 163, 6120, 6124	1881, 1887, 1902, 1915, 1922, 1931, 2126, 2153
<code>\@@_old_pgful@check@rerun</code> ..... 81, 85	<code>\@@_pred:n</code> .....
<code>\@@_old_vdots</code> ..... 1194, 3548	..... 124, 154, 2447, 4302, 4315, 4427, 4440
<code>\@@_open_x_final_dim:</code> .....	<code>\@@_provide_pgfsyspdfmark:</code> . 234, 243, 1443
..... 2999, 3051, 3085, 3243, 3292	<code>\@@_put_box_in_flow:</code> ..... 1576, 1934, 2119
<code>\@@_open_x_initial_dim:</code> .....	<code>\@@_put_box_in_flow_bis:nn</code> .... 1573, 2086
..... 2977, 3044, 3082, 3236, 3286	<code>\@@_put_box_in_flow_i:</code> ..... 1940, 1942
<code>\@@_open_y_final_dim:</code> 3123, 3175, 3241, 3291	<code>\@@_qpoint:n</code> ..... 256, 1945, 1947, 1959,
<code>\@@_open_y_initial_dim:</code> .....	1975, 2034, 2036, 2052, 2063, 2074, 2745,
..... 3105, 3172, 3233, 3285	2747, 2749, 2751, 2759, 2761, 2994, 3016,
<code>\l_@@_parallelize_diags_bool</code> .....	3045, 3052, 3091, 3093, 3107, 3125, 3181,
..... 489, 490, 588, 2554, 3247, 3295	3183, 3234, 3242, 3871, 3874, 4135, 4139,
<code>\@@_patch_node_for_cell:</code> ..... 1010, 1388	4155, 4157, 4325, 4327, 4329, 4450, 4452,
<code>\@@_patch_node_for_cell:n</code> 1008, 1034, 1037	4454, 4699, 4703, 4710, 4744, 4747, 4749,
<code>\@@_patch_preamble:n</code> ..... 1622,	4851, 4861, 5324, 5326, 5328, 5330, 5352,
1667, 1705, 1713, 1734, 1763, 1824, 1833,	5372, 5400, 5495, 5497, 5504, 5506, 5541,
1835, 1844, 1846, 1863, 1871, 1897, 1906, 1926	5543, 5547, 5552, 5554, 5558, 5601, 5603,
<code>\@@_patch_preamble_i:n</code> 1671, 1672, 1673, 1720	5605, 5612, 5616, 5618, 5737, 5739, 5742,
<code>\@@_patch_preamble_ii:nn</code> .....	5744, 5797, 5799, 5987, 5990, 6028, 6045, 6062
..... 1674, 1675, 1676, 1731	<code>\l_@@_radius_dim</code> ..... 480, 481, 1903,
<code>\@@_patch_preamble_iii:n</code> . 1677, 1736, 1744	2571, 3056, 3057, 3493, 4677, 4701, 4745, 4746
<code>\@@_patch_preamble_iii_i:n</code> .... 1739, 1741	<code>\l_@@_real_left_delim_dim</code> 2088, 2103, 2118
<code>\@@_patch_preamble_iv:nnn</code> .....	<code>\l_@@_real_right_delim_dim</code> 2089, 2115, 2121
..... 1678, 1679, 1680, 1766	<code>\@@_recreate_cell_nodes:</code> ..... 1351, 1413
<code>\@@_patch_preamble_ix:n</code> ..... 1911, 1929	<code>\g_@@_recreate_cell_nodes_bool</code> .....
<code>\@@_patch_preamble_v:nnnn</code> 1681, 1682, 1787	..... 496, 1351, 1380, 1387, 1392
<code>\@@_patch_preamble_vi:n</code> ..... 1683, 1809	<code>\@@_rectanglecolor</code> .. 1362, 3940, 3973, 3993
<code>\@@_patch_preamble_vii:nn</code> .....	<code>\@@_rectanglecolor:nnn</code> ... 3946, 3955, 3958
..... 1684, 1685, 1686, 1815	<code>\@@_renew_NC@rewrite@S:</code> 174, 178, 198, 1226
<code>\@@_patch_preamble_viii:nn</code> .....	<code>\@@_renew_dots:</code> ..... 1133, 1219
..... 1687, 1688, 1689, 1849	<code>\l_@@_renew_dots_bool</code> .....
<code>\@@_patch_preamble_viii_i:nnn</code> .. 1853, 1875	..... 589, 770, 1219, 6236, 6243
<code>\@@_patch_preamble_x:n</code> .....	<code>\@@_renew_matrix:</code> 765, 769, 5630, 6238, 6242
..... 1729, 1785, 1807, 1813, 1908, 1932	<code>\l_@@_respect_blocks_bool</code> 3999, 4014, 4042
<code>\@@_patch_preamble_xi:n</code> ..... 1697, 1900	<code>\@@_restore_iRow_jCol:</code> ..... 2630, 2675
<code>\@@_pgf_rect_node:nnn</code> .... 448, 1432, 5380	<code>\@@_revtex_array:</code> ..... 1049, 1060
<code>\@@_pgf_rect_node:nnnnn</code> .....	<code>\c_@@_revtex_bool</code> ..... 50, 52, 55, 57, 1059
..... 423, 4945, 4972, 5333, 5375, 6089	<code>\l_@@_right_delim_dim</code> .....
<code>\c_@@_pgfortikzpicture_tl</code> .....	..... 1292, 1296, 1302, 2120, 2405
..... 42, 46, 2659, 3863, 4691	<code>\g_@@_right_delim_tl</code> .. 1301, 1447, 1568,
<code>\@@_pgfpintanchor:n</code> ..... 6116, 6121	1574, 1635, 1823, 1857, 1858, 1879, 1884, 4719
<code>\@@_pgfpintanchor_i:nn</code> ..... 6124, 6126	<code>\l_@@_right_margin_dim</code> .....
<code>\@@_pgfpintanchor_ii:w</code> ..... 6127, 6135	..... 501, 598, 1495, 2406, 3190, 4715, 4937
<code>\@@_pgfpintanchor_iii:w</code> ..... 6148, 6150	<code>\@@_rotate:</code> ..... 1212, 3823
<code>\@@_picture_position:</code> .....	<code>\g_@@_rotate_bool</code> .....
..... 1331, 1340, 1348, 1434, 1435	..... 273, 935, 962, 1801, 2350,
<code>\g_@@_pos_of_blocks_seq</code> 313, 1472, 2549,	2395, 3823, 5072, 5096, 5101, 5161, 5177, 5320
2580, 2637, 3641, 4261, 4387, 4658, 5259, 5724	<code>\@@_rotate_cell_box:</code> .....
	..... 923, 962, 1801, 2350, 2395, 5320

\l_@@_rounded_corners_dim .....		\l_@@_submatrix_slim_bool .....	5860, 5955, 6459
327, 5200, 5304, 5487, 5488, 5523, 5569, 5626		\l_@@_submatrix_vlines_clist .....	343, 5864, 5882, 6035, 6037
\@@_roundedrectanglecolor .....	1363, 3949	\@@_succ:n .....	149,
\l_@@_row_max_int ....	319, 2782, 2924, 2941	153, 1082, 1088, 1114, 1761, 1829, 1840,	
\l_@@_row_min_int ....	318, 2850, 2922, 2939	1920, 1947, 2281, 2287, 2292, 2293, 2306,	
\g_@@_row_of_col_done_bool .....		2310, 2315, 2316, 2543, 3016, 3093, 3183,	
299, 1104, 1452, 2198		3242, 4094, 4139, 4155, 4298, 4329, 4367,	
\g_@@_row_total_int .....	260, 1223,	4423, 4454, 4490, 4513, 4663, 4665, 4667,	
1516, 1970, 2069, 2508, 2515, 2522, 2538,		4669, 4710, 4749, 4911, 4915, 4925, 4929,	
2695, 2980, 3002, 3738, 4805, 4819, 4846,		5328, 5330, 5372, 5504, 5506, 5742, 5744, 5799	
4941, 5252, 5338, 5356, 5804, 5943, 5957, 6438		\l_@@_suffix_tl .....	4873, 4884,
\@@_rowcolor .....	1364, 3910	4894, 4897, 4946, 4954, 4955, 4973, 4981, 4982	
\@@_rowcolor:n .....	3916, 3919	\c_@@_table_collect_begin_tl .	205, 225, 227
\@@_rowcolor_tabular .....	1146, 4203	\c_@@_table_print_tl .....	207, 228, 229
\@@_rowcolors .....	1365, 4006	\l_@@_tabular_width_dim .....	
\@@_rowcolors_i:nnnn .....	4050, 4085	271, 1065, 1067, 1661, 2476	
\l_@@_rowcolors_restart_bool ...	4002, 4033	\l_@@_tabularnote_tl	357, 825, 850, 1991, 2000
\g_@@_rows_seq .	2148, 2150, 2152, 2154, 2156	\g_@@_tabularnotes_seq .....	
\l_@@_rows_tl .....		356, 396, 2006, 2012, 2028	
3921, 3936, 3966, 4052, 4116, 4141		\@@_test_hline_in_block:nnnn .....	
\l_@@_rules_color_tl .....		4388, 4390, 4516	
297, 531, 1485, 1486, 6018, 6019		\@@_test_hline_in_stroken_block:nnnn .	
\@@_set_CT@arc@: .....	157, 1486, 6019	4392, 4538	
\@@_set_CT@arc@i: .....	158, 159	\@@_test_if_cell_in_a_block:nn .....	
\@@_set_CT@arc@ii: .....	158, 161	4597, 4615, 4633, 4653	
\@@_set_final_coords: .....	2950, 2975	\@@_test_if_cell_in_block:nnnnnnn ...	
\@@_set_final_coords_from_anchor:n ..		4659, 4661	
2966, 3055, 3086, 3167, 3176, 3246, 3294		\@@_test_if_math_mode: ....	274, 1458, 2428
\@@_set_initial_coords: .....	2945, 2964	\@@_test_in_corner_h: .....	4393, 4421
\@@_set_initial_coords_from_anchor:n .		\@@_test_in_corner_v: .....	4268, 4296
2955, 3048, 3083, 3166, 3173, 3238, 3288		\@@_test_vline_in_block:nnnn .....	
\@@_set_size:n .....	5648, 5657	4262, 4264, 4527	
\c_@@_siunitx_loaded_bool	164, 168, 173, 217	\@@_test_vline_in_stroken_block:nnnn .	
\l_@@_small_bool .....	763, 810, 816,	4266, 4549	
838, 869, 1161, 1817, 1851, 2332, 2378, 2569		\l_@@_the_array_box ..	1288, 1304, 1985, 1986
\@@_standard_cline .....	120, 1198	\c_@@_tikz_loaded_bool .....	
\@@_standard_cline:w .....	120, 121	27, 41, 1353, 2591, 4728	
\l_@@_standard_cline_bool ..	473, 543, 1197	\@@_true_c: .....	190, 206, 1683
\c_@@_standard_tl	483, 484, 3320, 4721, 4751	\l_@@_type_of_col_tl ..	808, 809, 2457, 2459
\g_@@_static_num_of_col_int .....		\c_@@_types_of_matrix_seq .....	
323, 1585, 1623, 5240, 6343, 6355, 6536		6298, 6299, 6304, 6308	
\l_@@_stop_loop_bool .....	2776, 2777,	\@@_update_for_first_and_last_row: ..	
2809, 2822, 2831, 2844, 2845, 2877, 2890, 2899		906, 970, 1242, 2352, 2397	
\@@_store_in_tmpb_tl .....	5465, 5467	\@@_use_arraybox_with_notes: ...	1530, 2047
\@@_stroke_block:nnn .....	5266, 5469	\@@_use_arraybox_with_notes_b: .	1527, 2031
\@@_stroke_borders_block:nnn ...	5288, 5565	\@@_use_arraybox_with_notes_c: .....	
\@@_stroke_horizontal:n ..	5593, 5595, 5610	1528, 1559, 1983, 2045, 2084	
\@@_stroke_vertical:n ....	5589, 5591, 5599	\@@_vdottedline:n .....	1905, 4724
\@@_sub_matrix:nnnnn .....	5927, 5929	\@@_vdottedline_i:n .....	4731, 4736, 4740
\@@_sub_matrix_i:nn .....	5979, 5985	\@@_vline:nn .....	1761, 4245, 4371
\l_@@_submatrix_extra_height_dim ....		\@@_vline_i:nn .....	2585, 4250, 4254
337, 5850, 6013		\@@_vline_i_complete:nn .....	2585, 4361
\l_@@_submatrix_hlines_clist .....		\@@_vline_ii:nnnn ...	4277, 4288, 4321, 4362
342, 5862, 5880, 6052, 6054		\l_@@_vlines_clist .....	
\l_@@_submatrix_left_xshift_dim .....		341, 573, 585, 614, 1608,	
338, 5852, 6065, 6098		1614, 1644, 1656, 1913, 1920, 2589, 4369, 4370	
\l_@@_submatrix_name_str .....		\l_@@_vpos_of_block_tl .....	332, 333,
5895, 5947, 6087, 6089, 6101, 6103, 6111, 6114		4993, 4995, 5077, 5087, 5165, 5182, 5216, 5218	
\g_@@_submatrix_names_seq .....		\@@_w: .....	1601, 1681
317, 2607, 5892, 5896, 6486		\g_@@_width_first_col_dim .....	
\l_@@_submatrix_right_xshift_dim ....		311, 1451, 1521, 2193, 2353, 2354	
339, 5854, 6074, 6108			
\g_@@_submatrix_seq ...	322, 1238, 2926, 5918		



<code>\g_@@_width_last_col_dim</code> .....		<code>\arraybackslash</code> .....	1774
..... 310, 1450, 1580, 2302, 2398, 2399		<code>\arraycolor</code> .....	1366, 6294
<code>\@@_write_aux_for_cell_nodes:</code> ..	2628, 2690	<code>\arraycolsep</code> .....	597, 599, 601, 1064, 1164, 1295, 1296, 1558, 1562, 4707, 4714
<code>\l_@@_x_final_dim</code> .....	305, 2952, 3001, 3010, 3011, 3014, 3017, 3018, 3169, 3185, 3193, 3197, 3201, 3203, 3208, 3210, 3244, 3253, 3261, 3301, 3309, 3348, 3363, 3372, 3406, 3458, 3474, 3875, 4711, 4720, 4746, 5954, 5970, 5971, 5977, 6074, 6091, 6108	<code>\arrayrulecolor</code> .....	95
<code>\l_@@_x_initial_dim</code> .....	303, 2947, 2979, 2988, 2989, 2992, 2995, 2996, 3169, 3184, 3185, 3192, 3197, 3201, 3203, 3205, 3208, 3210, 3235, 3253, 3261, 3301, 3309, 3345, 3362, 3372, 3406, 3458, 3472, 3474, 3492, 3494, 3872, 4704, 4718, 4745, 5953, 5963, 5964, 5974, 6065, 6090, 6098	<code>\arrayrulewidth</code> .....	128, 133, 145, 533, 897, 1081, 1083, 1089, 1120, 1611, 1617, 1704, 1754, 1916, 1923, 2039, 2078, 2205, 2207, 2213, 2223, 2225, 2231, 2254, 2256, 2262, 2280, 2282, 2288, 4137, 4138, 4140, 4156, 4158, 4337, 4338, 4340, 4351, 4357, 4463, 4474, 4480, 4509, 4786, 5473, 5528, 5553, 5555, 5567, 5822, 6013, 6017
<code>\l_@@_xdots_color_tl</code> 506, 520, 3035, 3074, 3155, 3156, 3224, 3276, 3328, 3742, 3817, 3834		<code>\arraystretch</code> .....	1163, 3109, 3127, 5069, 5158, 5174, 5989, 5992
<code>\l_@@_xdots_down_tl</code> ... 524, 3335, 3356, 3391		<code>\AtBeginDocument</code> .....	23, 28, 73, 89, 165, 171, 363, 477, 479, 481, 493, 677, 693, 2655, 3499, 3672, 3748, 3826, 3859, 4685
<code>\l_@@_xdots_line_style_tl</code> .....	482, 484, 516, 3320, 3328, 4721, 4751	<code>\AutoNiceMatrix</code> .....	5693
<code>\l_@@_xdots_shorten_dim</code> .....	478, 479, 522, 2573, 3342, 3343, 3432, 3443, 3451	<code>\AutoNiceMatrixWithDelims</code> ...	5653, 5685, 5697
<code>\l_@@_xdots_up_tl</code> .... 525, 3334, 3355, 3381		<b>B</b>	
<code>\l_@@_y_final_dim</code> .....	306, 2953, 3053, 3057, 3095, 3099, 3101, 3126, 3136, 3137, 3255, 3258, 3303, 3306, 3348, 3363, 3371, 3408, 3463, 3482, 3876, 4702, 4750, 5800, 5822, 5837, 5991, 6006, 6007, 6012, 6030, 6047, 6091, 6099, 6109	<code>\baselineskip</code> .....	98, 105
<code>\l_@@_y_initial_dim</code> .....	304, 2948, 3046, 3056, 3094, 3095, 3099, 3101, 3108, 3118, 3119, 3255, 3260, 3303, 3308, 3345, 3362, 3371, 3408, 3463, 3480, 3482, 3492, 3495, 3873, 4700, 4701, 4702, 4748, 5798, 5822, 5837, 5988, 5999, 6000, 6012, 6029, 6046, 6090, 6099, 6109	<code>\bgroup</code> .....	1445
<code>\</code> .....	2139, 2161, 5664, 5670, 5676, 6214, 6215, 6258, 6267, 6295, 6355, 6360, 6365, 6370, 6375, 6413, 6418, 6424, 6431, 6438, 6444, 6445, 6460, 6468, 6474, 6480, 6481, 6492, 6504, 6511, 6517, 6524, 6531, 6540, 6546, 6553, 6560, 6566, 6572, 6584, 6588, 6593, 6599, 6608, 6609, 6623, 6624, 6640, 6644, 6645, 6663, 6664, 6706, 6707, 6758, 6759, 6809, 6810, 6863, 6864	<code>\bigskip</code> .....	1487, 2735
<code>\{</code> .....	290, 1686, 1830, 1841, 1867, 2435, 5692, 6070, 6473, 6553, 6706, 6809	<code>\Block</code> .....	1211, 6510, 6558, 6599
<code>\}</code> .....	290, 1689, 1830, 1841, 1852, 2435, 5692, 6079, 6473, 6553, 6706, 6809	<code>\BNiceMatrix</code> .....	5645
<code>\ </code> .....	2437, 5691	<code>\bNiceMatrix</code> .....	5642
<code>\_</code> ..	6294, 6322, 6327, 6334, 6342, 6343, 6354, 6355, 6412, 6437, 6438, 6451, 6457, 6461, 6464, 6467, 6479, 6485, 6497, 6535, 6536, 6537, 6545, 6551, 6558, 6571, 6578, 6579, 6587	<code>\Body</code> .....	1312
<b>A</b>		bool commands:	
<code>\A</code> .....	5890	<code>\bool_do_until:Nn</code> .....	2777, 2845
<code>\aboverulesep</code> .....	2022	<code>\bool_gset_false:N</code> .....	935, 981, 982, 1103, 1227, 1380, 1452, 2364, 2411, 4525, 4536, 4547, 4558, 5101
<code>\addtocounter</code> .....	413	<code>\bool_gset_true:N</code> .....	2198, 2328, 2373, 2464, 3517, 3533, 3549, 3573, 3596, 3607, 3823, 4260, 4386
<code>\alpha</code> .....	359	<code>\bool_if:N</code> .....	156, 217, 679, 684, 695, 700, 866, 869, 962, 1077, 1104, 1151, 1161, 1218, 1219, 1239, 1289, 1351, 1353, 1382, 1385, 1387, 1444, 1459, 1469, 1500, 1578, 1583, 1594, 1599, 1625, 1801, 1817, 1851, 2018, 2184, 2201, 2219, 2237, 2250, 2276, 2297, 2303, 2332, 2350, 2378, 2395, 2485, 2487, 2506, 2509, 2528, 2554, 2569, 2591, 2608, 2611, 2628, 3098, 3100, 3247, 3295, 3515, 3531, 3547, 3571, 3594, 4606, 4624, 4642, 4769, 4779, 4801, 5072, 5096, 5161, 5177, 5274, 5320, 5378, 5704, 6270, 6280, 6313, 6459
<code>\anchor</code> .....	2687, 2688	<code>\bool_if:nTF</code> .....	173, 365, 392, 1040, 1510, 2931, 3848, 4100, 4795, 5428, 5801, 5811, 5813, 5826, 5832, 5841
		<code>\bool_lazy_all:nTF</code> .....	1640, 1652, 2578, 4518, 4529, 4540, 4551
		<code>\bool_lazy_and:nnTF</code> .....	1708, 2240, 2333, 2540, 3087, 3354, 3656, 4013, 4041, 4111, 4331, 4456, 5491, 6039, 6056
		<code>\bool_lazy_or:nnTF</code> .....	513, 974, 1032, 1633, 1968, 1989, 2067, 2381, 3164, 3412, 4092, 4124, 4128, 4178, 4182, 4598, 4616, 4634, 5013, 5018, 5038, 5942
		<code>\bool_lazy_or_p:nn</code> .....	2336

`\bool_not_p:n` ..... 1643, 1645, 1655, 1657, 2242, 2542  
`\bool_set:Nn` ..... 3168, 4035  
`\c_false_bool` ..... 1862, 1870, 1883, 1889, 1895, 3511, 3527, 3543  
`\g_tmpa_bool` ..... 4260, 4269, 4303, 4311, 4316, 4386, 4394, 4428, 4436, 4441, 4525, 4536, 4547, 4558  
`\l_tmpb_bool` ... 4603, 4617, 4635, 4657, 4670  
`\c_true_bool` ..... 1829, 1840  
box commands:  
`\box_clear_new:N` ..... 1153, 1288  
`\box_dp:N` ..... 891, 911, 948, 967, 1022, 1179, 1188, 1247, 1779, 1937, 2097, 2110, 3127, 5131, 5992  
`\box_gclear_new:N` ..... 5060  
`\box_grotate:Nn` ..... 5098  
`\box_ht:N` ..... 892, 913, 919, 931, 954, 965, 1014, 1181, 1183, 1186, 1245, 1775, 1936, 2097, 2110, 3109, 5122, 5989  
`\box_move_down:nn` ..... 1022  
`\box_move_up:nn` ..... 64, 66, 68, 1014, 1980, 2045, 2084  
`\box_rotate:Nn` ..... 925  
`\box_set_dp:Nn` ..... 947, 966, 1937  
`\box_set_ht:Nn` ..... 953, 964, 1936  
`\box_set_wd:Nn` ..... 941  
`\box_use:N` ..... 416, 932, 1021  
`\box_use_drop:N` ... 972, 978, 997, 1803, 1939, 1980, 1981, 1986, 2359, 5141, 5423, 5457  
`\box_wd:N` ..... 417, 942, 969, 976, 1035, 1300, 1302, 1985, 2104, 2116, 2354, 2357, 2399, 2403, 5110, 5711  
`\l_tmpa_box` ..... 399, 416, 417, 1299, 1300, 1301, 1302, 1545, 1936, 1937, 1939, 1980, 1981, 2097, 2110  
`\l_tmpb_box` ..... 2090, 2104, 2105, 2116

## C

`\c` ..... 233, 1629  
`\Cdots` ..... 1201, 3522, 3525  
`\cdots` ..... 1136, 1193  
`\cellcolor` ..... 1145, 1361, 4198  
`\chessboardcolors` ..... 1368  
`\cline` ..... 148, 1198, 1199  
clist commands:  
`\clist_clear:N` ..... 4171  
`\clist_if_empty:Ntf` ..... 4267, 4393, 5284  
`\clist_if_in:NnTF` ..... 1112, 1614, 1920, 4370, 4493, 5588, 5590, 5592, 5594, 5613  
`\clist_if_in:nnTF` ..... 5574  
`\clist_map_inline:Nn` ..... 4118, 4141, 4172, 4563, 5572, 6037, 6054  
`\clist_map_inline:nn` ... 2452, 3972, 4026  
`\clist_new:N` ... 326, 340, 341, 342, 343, 491  
`\clist_put_right:Nn` ..... 4189  
`\clist_set:Nn` ..... 585, 586, 613, 614, 615  
`\clist_set_eq:NN` ..... 4170  
`\l_tmpa_clist` ..... 4170, 4172  
`\CodeAfter` ..... 861, 1215, 2142, 2145, 2327, 2372, 2606, 6611  
`\CodeBefore` ..... 1441, 6294  
`\color` ..... 99, 106, 160, 162, 1549, 1567, 3029, 3032, 3035, 3068, 3071, 3074,

3149, 3152, 3156, 3224, 3276, 3736, 3739, 3742, 3811, 3814, 3817, 3834, 3898, 4059, 4060, 4071, 4072, 5067, 5202, 5831, 6171, 6195  
`\colorlet` ..... 282, 283, 876, 883, 2342, 2386  
`\columncolor` ..... 1147, 1367, 2617, 4221  
`\cr` ..... 132, 150, 2321  
`\crrcr` ..... 2178

cs commands:

`\cs_generate_variant:Nn` ..... 58, 152, 3351, 3891, 4791, 4792  
`\cs_gset:Npn` ..... 99, 106, 2493, 2500, 2514, 2521, 4784  
`\cs_gset_eq:NN` ... 230, 243, 1172, 1461, 2631  
`\cs_if_exist:Ntf` ..... 57, 176, 1154, 1157, 1255, 1262, 1273, 1280, 1423, 1462, 1465, 2677, 2678, 2699, 2812, 2825, 2880, 2893, 2982, 3004, 3112, 3130, 3704, 3722, 3779, 3797, 4771, 4824, 5340, 5358, 5806, 5959, 5966, 5995, 6002  
`\cs_if_exist_p:N` ..... 514, 4016, 4044, 4600, 4619, 4637  
`\cs_if_free:Ntf` ..... 239, 3024, 3063, 3144, 3219, 3271  
`\cs_if_free_p:N` ..... 3850, 3852  
`\cs_new_protected:Npx` ... 2657, 3861, 4687  
`\cs_set:Nn` ..... 665, 668, 671  
`\cs_set:Npn` ..... 95, 96, 102, 103, 108, 120, 121, 135, 137, 138, 160, 162, 362, 1128, 2771, 2833, 2901, 3746, 3821, 4497, 4498, 4504, 4505, 5069, 5158, 5174  
`\cs_set_nopar:Npn` ..... 1054, 1066, 1163, 1166, 4966, 4967  
`\cs_set_nopar:Npx` ..... 1067  
`\cs_set_protected:Npn` ..... 5682  
`\cs_set_protected_nopar:Npn` ... 5048, 5307

## D

`\Ddots` ..... 1203, 3555, 3556, 3561, 3562  
`\ddots` ..... 1138, 1195  
`\diagbox` ..... 1216, 5048, 5307  
dim commands:  
`\dim_add:Nn` ..... 4935  
`\dim_compare:nNnTF` ..... 98, 105, 939, 945, 951, 1661, 2238, 2403, 2992, 3014, 3203, 4848, 4858, 5350, 5370, 5711  
`\dim_gzero:N` ..... 943, 949, 955  
`\dim_max:nn` ..... 4837, 4841  
`\dim_min:nn` ..... 4829, 4833  
`\dim_ratio:nn` ..... 3262, 3310, 3426, 3431, 3442, 3450, 3459, 3464, 3475, 3483  
`\dim_set:Nn` ... 4810, 4817, 4828, 4832, 4836, 4840, 4852, 4853, 4862, 4863, 4907, 4920  
`\dim_set_eq:NN` ..... 4808, 4815, 4915, 4929  
`\dim_sub:Nn` ..... 4932  
`\dim_use:N` ..... 4849, 4859, 4910, 4911, 4923, 4924, 4947, 4948, 4949, 4950, 4974, 4975, 4976, 4977  
`\dim_zero_new:N` ..... 4807, 4809, 4814, 4816  
`\c_max_dim` . 2979, 2992, 3001, 3014, 4808, 4810, 4815, 4817, 4849, 4859, 5337, 5350, 5355, 5370, 5802, 5803, 5953, 5954, 5974, 5977  
`\dotfill` ..... 1214, 5700  
`\dots` ..... 1140  
`\doublerulesep` 1755, 4340, 4352, 4463, 4475, 4510

<code>\doublerulesepcolor</code>	102
<code>\draw</code>	3339
<b>E</b>	
<code>\egroup</code>	1593
else commands:	
<code>\else:</code>	276
<code>\endarray</code>	2130, 2158
<code>\endBNiceMatrix</code>	5646
<code>\endbNiceMatrix</code>	5643
<code>\endNiceArray</code>	2472, 2481
<code>\endNiceArrayWithDelims</code>	2421, 2431
<code>\endpgfscope</code>	2713, 2726, 3396, 5757
<code>\endpNiceMatrix</code>	5634
<code>\endsavenotes</code>	1594
<code>\endtabularnotes</code>	2013
<code>\endVNiceMatrix</code>	5640
<code>\endvNiceMatrix</code>	5637
<code>\enskip</code>	1827, 1860, 1868, 1881, 1887
<code>\ensuremath</code>	3516, 3532, 3548, 3572, 3595
<code>\everycr</code>	131, 150, 1176
exp commands:	
<code>\exp_after:wN</code>	182, 202, 1486, 1550, 1568, 1622, 6019
<code>\exp_args:Ne</code>	3617
<code>\exp_args:NNc</code>	4773
<code>\exp_args:NNe</code>	3613
<code>\exp_args:Nne</code>	2459
<code>\exp_args:NNV</code>	2150, 3503, 3519, 3535, 3551, 3575, 3676, 3752, 3830
<code>\exp_args:NNv</code>	1478
<code>\exp_args:NNx</code>	1111, 1919
<code>\exp_args:No</code>	3327
<code>\exp_args:NV</code>	1603, 2126, 2153, 2155, 5482
<code>\exp_args:Nxx</code>	5041, 5042
<code>\exp_last_unbraced:NV</code>	1383, 2609, 5296
<code>\exp_not:N</code>	42, 43, 46, 47, 1704, 1748, 4209, 4221, 4689, 4690, 5077, 5087, 5165, 5182, 5299, 6124
<code>\exp_not:n</code>	1046, 2623, 3686, 3762, 4198, 4209, 4211, 4222, 5057, 5139, 5151, 5152, 5267, 5279, 5289, 5301, 5316, 5721, 5722
<code>\ExplSyntaxOff</code>	241, 2504, 2525, 2551, 2625, 4587, 4788
<code>\ExplSyntaxOn</code>	238, 2490, 2511, 2530, 2618, 4580, 4781
<code>\extrarowheight</code>	3109, 5070, 5159, 5175, 5989
<b>F</b>	
<code>\fi</code>	110, 1605, 4497, 4514
fi commands:	
<code>\fi:</code>	278
<code>\five</code>	2682, 2687
flag commands:	
<code>\flag_clear_new:n</code>	6117
<code>\flag_height:n</code>	6142
<code>\flag_raise:n</code>	6141
<code>\fontdimen</code>	1976
fp commands:	
<code>\fp_eval:n</code>	3367
<code>\fp_to_dim:n</code>	3402
<code>\futurelet</code>	115

<b>G</b>	
<code>\globaldefs</code>	1589, 5245
group commands:	
<code>\group_insert_after:N</code>	5705, 5706, 5708, 5709
<b>H</b>	
<code>\halign</code>	1190
<code>\hbox</code>	1075, 1554, 2045, 2084, 2203, 2221, 2248, 2252, 2278, 2708, 2721
hbox commands:	
<code>\hbox:n</code>	64, 66, 69
<code>\hbox_gset:Nn</code>	5062
<code>\hbox_overlap_left:n</code>	1015, 1023, 2182, 2355
<code>\hbox_overlap_right:n</code>	416, 2299, 2401
<code>\hbox_set:Nn</code>	399, 1012, 1299, 1301, 1545, 2090, 2105, 5319
<code>\hbox_set:Nw</code>	865, 1304, 1792, 2330, 2376
<code>\hbox_set_end:</code>	961, 1497, 1800, 2349, 2394
<code>\hbox_to_wd:nn</code>	441, 466
<code>\Hdotsfor</code>	1208, 6322, 6497
<code>\hdotsfor</code>	1141
<code>\hdottedline</code>	1205
<code>\heavyrulewidth</code>	2023
<code>\hfil</code>	1682
<code>\hfill</code>	128, 145
<code>\Hline</code>	1206, 4500
<code>\hline</code>	108
<code>\hrule</code>	112, 128, 145, 1120, 2023, 5836, 6176, 6200
<code>\hskip</code>	111
<code>\Hspace</code>	1207
<code>\hspace</code>	3608
<code>\hss</code>	1682
<b>I</b>	
<code>\ialign</code>	1072, 1166, 1189, 5224
<code>\iddots</code>	1204, 3579, 3580, 3585, 3586
<code>\iddots</code>	59, 1139, 1196
if commands:	
<code>\if_mode_math:</code>	276
<code>\IfBooleanTF</code>	1488
<code>\ifnum</code>	110, 4497, 4514
<code>\ifstandalone</code>	1465
int commands:	
<code>\int_case:nnTF</code>	3553, 3559, 3577, 3583
<code>\int_compare:nNnTF</code>	123, 124, 140, 863, 864, 873, 880, 916, 926, 1117, 1119, 1240, 1250, 1269, 1326, 1328, 1498, 1502, 1513, 1517, 1518, 1585, 2001, 2040, 2079, 2151, 2179, 2379, 2787, 2794, 2798, 2800, 2855, 2862, 2866, 2868, 3031, 3070, 3151, 3186, 3188, 3636, 3650, 3738, 3813, 4087, 4132, 4150, 4186, 4217, 4234, 4235, 4242, 4243, 4247, 4663, 4665, 4667, 4669, 5066, 5103, 5115, 5397, 5426, 5430, 5500, 5502, 5660, 5662, 5664, 5668, 5670, 5672, 5674, 5676, 6023, 6025
<code>\int_compare_p:n</code>	2933, 2934, 2935, 2936, 4102, 4104, 5492, 5493
<code>\int_do_until:nNnn</code>	4038
<code>\int_gadd:Nn</code>	3649
<code>\int_gincr:N</code>	862, 889, 1468, 1728, 1784, 1806, 1812, 2274, 2374, 3249, 3297, 4766, 5047
<code>\int_if_even:nTF</code>	3981, 6142
<code>\int_if_odd_p:n</code>	4035
<code>\int_incr:N</code>	395, 1738



`\int_min:nn` ..... 2745, 2747, 2749, 2751, 2759, 2761, 2941, 2942  
`\int_step_inline:nn` ..... 2743, 3977, 3979, 6036, 6053  
`\int_step_inline:nnnn` 4595, 4613, 4628, 4631  
`\c_zero_int` ..... 1818, 3884, 4093  
iow commands:  
`\iow_now:Nn` ..... 76, 236, 2530, 2531, 2533, 2551, 2618, 2619, 2625, 4580, 4581, 4587  
`\iow_shipout:Nn` ..... 2490, 2491, 2498, 2504, 2511, 2512, 2519, 2525, 4781, 4782, 4788  
`\item` ..... 2006, 2012

## K

`\kern` ..... 69  
keys commands:  
`\keys_define:nn` ..... 509, 529, 536, 608, 655, 707, 714, 758, 800, 814, 831, 844, 1390, 3599, 3988, 3997, 4755, 4985, 5194, 5517, 5623, 5763, 5768, 5848, 5869, 5878, 6232  
`\l_keys_key_str` ..... 6214, 6396, 6402, 6490, 6593, 6598, 6608, 6623, 6644, 6662, 6705, 6757, 6808  
`\keys_set:nn` ..... 187, 539, 541, 555, 789, 792, 799, 1394, 1402, 1482, 1483, 2458, 2468, 2477, 2634, 3034, 3073, 3154, 3223, 3275, 3741, 3816, 3833, 3992, 4011, 4768, 5261, 5770, 5774, 5901, 5948  
`\keys_set_known:nn` ..... 3565, 3589, 5030, 5474, 5529, 5568  
`\l_keys_value_tl` .... 6443, 6519, 6526, 6858

## L

`\Ldots` ..... 1200, 3506, 3509  
`\ldots` ..... 1135, 1192  
`\leaders` ..... 128, 145  
`\left` ..... 1550, 2093, 2108, 5832, 6172, 6196  
legacy commands:  
`\legacy_if:nTF` ..... 639  
`\line` ..... 2603, 6473, 6619

## M

`\makebox` ..... 1803  
`\mathinner` ..... 61  
mode commands:  
`\mode_leave_vertical:` ..... 1457, 1773  
msg commands:  
`\msg_error:nn` ..... 12  
`\msg_error:nnn` ..... 13  
`\msg_error:nnnn` ..... 14, 5242, 5249, 5253  
`\msg_fatal:nn` ..... 15  
`\msg_fatal:nnn` ..... 16  
`\msg_new:nnn` ..... 17  
`\msg_new:nnnn` ..... 18  
`\msg_redirect_name:nnn` ..... 20  
`\multicolumn` ..... 1210, 3610, 3662, 3668, 3689  
`\multispan` ..... 124, 125, 141, 142  
`\myfiledate` ..... 6  
`\myfileversion` ..... 7

## N

`\newcolumntype` ..... 248, 249, 250  
`\newcounter` ..... 355

`\NewDocumentCommand` ..... 367, 390, 798, 1400, 2633, 3503, 3519, 3535, 3551, 3575, 3676, 3752, 3830, 3910, 3925, 3940, 3949, 3970, 3975, 3990, 4006, 4192, 4203, 4215, 5462, 5653, 5693, 5908, 5924  
`\NewDocumentEnvironment` ..... 1440, 2123, 2132, 2414, 2424, 2454, 2465, 2473, 4764  
`\NewExpandableDocumentCommand` ..... 254, 5000  
`\newlist` ..... 371, 382  
`\NiceArray` ..... 2470, 2479  
`\NiceArrayWithDelims` ..... 2419, 2429  
nicematrix commands:

`\g_nicematrix_code_after_tl` .. 292, 648, 2147, 2609, 2612, 5264, 5276, 5286, 5781, 5788  
`\g_nicematrix_code_before_tl` ..... 1234, 2614, 2623, 4196, 4207, 4219, 5297  
`\NiceMatrixLastEnv` ..... 254  
`\NiceMatrixOptions` ..... 798, 6663, 6863  
`\NiceMatrixoptions` ..... 6523  
`\noalign` 98, 105, 110, 133, 1099, 1172, 4497, 4677  
`\nobreak` ..... 385  
`\normalbaselines` ..... 1160  
`\NotEmpty` ..... 1217  
`\nulldelimiterspace` .... 2104, 2116, 5817, 6015  
`\nullfont` .... 5828, 5835, 6168, 6175, 6192, 6199  
`\numexpr` ..... 153, 154

## O

`\omit` ..... 123, 2181, 2197, 2273, 5778  
`\OnlyMainNiceMatrix` ..... 1213, 4226

## P

`\par` ..... 2000, 2008  
peek commands:  
`\peek_meaning:NTF` ..... 158, 397, 1129  
`\peek_meaning_ignore_spaces:NTF` 2125, 4500  
`\peek_meaning_remove_ignore_spaces:NTF` 148  
`\peek_remove_spaces:n` ..... 3634, 4194, 4205, 5002, 5910, 5926  
`\pgfdeclareshape` ..... 2680  
`\pgfextracty` ..... 5400  
`\pgfgetlastxy` ..... 459  
`\pgfpathcircle` ..... 3491  
`\pgfpathlineto` .... 4348, 4354, 4471, 4477, 5549, 5560, 5607, 5620, 5746, 6030, 6047, 6081  
`\pgfpathmoveto` .... 4347, 4353, 4470, 4476, 5548, 5559, 5606, 5619, 5741, 6029, 6046, 6072  
`\pgfpathrectanglecorners` 4160, 4341, 4464, 5508  
`\pgfpointhead` ..... 457  
`\pgfpointheadanchor` ..... 163, 257, 2703, 2716, 2957, 2968, 2985, 3007, 3115, 3133, 4827, 4835, 5345, 5363, 5382, 5384, 5401, 5435, 5809, 5962, 5969, 5998, 6005, 6116, 6120  
`\pgfpointdiff` ..... 458, 1340, 1348, 1434, 1435  
`\pgfpointlineatime` ..... 3361  
`\pgfpointorigin` ..... 2188, 2311, 2688  
`\pgfpointscale` ..... 457  
`\pgfpointshapeborder` ..... 3871, 3874  
`\pgfrememberpicturepositiononpagetrue` ... 894, 988, 1087, 2187, 2211, 2229, 2260, 2286, 2309, 2666, 2693, 2742, 3318, 3870, 4323, 4448, 4698, 4743, 4870, 4880, 4891, 5322, 5476, 5537, 5584, 5736, 5795, 5950  
`\pgfscope` ..... 2701, 2714, 3358, 5753

`\pgfset` 426, 451, 989, 5412, 5446, 5752, 5816, 5952  
`\pgfsetbaseline` ..... 987  
`\pgfsetcornersarced` ..... 4159, 5484  
`\pgfsetlinewidth` .....  
..... 4357, 4480, 5511, 5540, 5587, 6017  
`\pgfsetrectcap` ..... 4358, 4481  
`\pgfsetroundcap` ..... 5749  
`\pgfsetstrokecolor` ..... 5482  
`\pgfsyspdfmark` ..... 239, 240  
`\pgftransformrotate` ..... 3365  
`\pgftransformshift` 432, 457, 2702, 2715, 2753,  
3359, 5411, 5433, 5754, 5758, 5818, 6095, 6105  
`\pgfusepath` .....  
..... 3385, 3395, 3901, 4063, 4075, 4344, 5512  
`\pgfusepathqfill` ..... 3497, 4467  
`\pgfusepathqstroke` ..... 4359, 4482,  
5550, 5561, 5608, 5621, 5750, 6031, 6048, 6082  
`\phantom` ..... 3516, 3532, 3548, 3572, 3595  
`\pNiceMatrix` ..... 5633  
prg commands:  
`\prg_do_nothing:` .....  
..... 174, 221, 230, 243, 526, 1172, 2606  
`\prg_new_conditional:Nnn` ..... 4090, 4098  
`\prg_replicate:nn` ..... 405, 2269,  
2270, 3689, 4349, 4472, 5663, 5666, 5669, 5675  
`\prg_return_false:` ..... 4095, 4107  
`\prg_return_true:` ..... 4096, 4106  
`\ProcessKeysOptions` ..... 6251  
`\ProvideDocumentCommand` ..... 59  
`\ProvidesExplPackage` ..... 4

## Q

`\quad` ..... 386  
quark commands:  
`\q_stop` 120, 121, 137, 138, 159, 161, 1383,  
1405, 1486, 1622, 1693, 1855, 1877, 2142,  
2145, 3824, 3838, 3839, 3905, 3960, 3965,  
4030, 4122, 4123, 4145, 4146, 4176, 4177,  
4964, 4971, 5005, 5006, 5010, 5296, 5467,  
5490, 5499, 5530, 5533, 5577, 5580, 5648,  
5657, 5912, 5917, 5936, 5939, 6019, 6127, 6135

## R

`\rectanglecolor` ..... 1362, 2616, 4209  
`\refstepcounter` ..... 414  
regex commands:  
`\regex_const:Nn` ..... 233  
`\regex_match:nnTF` ..... 5890  
`\regex_replace_all:NnN` ..... 1627  
`\relax` ..... 153, 154  
`\renewcommand` ..... 180, 200  
`\RenewDocumentEnvironment` .....  
..... 5632, 5635, 5638, 5641, 5644  
`\RequirePackage` ..... 1, 3, 9, 10, 11  
`\right` ..... 1568, 2100, 2112, 5841, 6180, 6204  
`\rotate` ..... 1212  
`\roundedrectanglecolor` ..... 1363, 5299  
`\rowcolor` ..... 1146, 1364  
`\rowcolors` ..... 1365

## S

`\savedanchor` ..... 2682  
`\savenotes` ..... 1444

scan commands:

`\scan_stop:` ..... 2610  
`\scriptstyle` .....  
..... 869, 2332, 2378, 3346, 3347, 3381, 3391

seq commands:

`\seq_clear:N` ..... 1620  
`\seq_clear_new:N` ..... 2532, 4562  
`\seq_count:N` ..... 2152, 3887  
`\seq_gclear:N` .....  
..... 1238, 1471, 1472, 1473, 1474, 2028, 2607  
`\seq_gclear_new:N` 1220, 1221, 1379, 2148, 2164  
`\seq_gpop_left:Nn` ..... 2154, 2166  
`\seq_gput_left:Nn` .... 644, 3638, 3640, 5259  
`\seq_gput_right:Nn` .... 396, 1701, 2912,  
3641, 3886, 5136, 5148, 5271, 5724, 5896, 5918  
`\seq_gset_from_clist:Nn` .. 2535, 2547, 4583  
`\seq_gset_map_x:Nnn` ..... 2637  
`\seq_gset_split:Nnn` ..... 2150, 2165  
`\seq_if_empty:NTF` ..... 1988, 4578  
`\seq_if_empty_p:N` ... 2580, 2581, 2582, 4112  
`\seq_if_exist:NTF` ..... 1370, 1407  
`\seq_if_in:NnTF` .....  
..... 642, 4152, 4300, 4306, 4313, 4425,  
4431, 4438, 4830, 4838, 5343, 5361, 5892, 6308  
`\seq_item:Nn` ..... 1322, 1325,  
1335, 1336, 1343, 1344, 1416, 1417, 1420, 1421  
`\seq_map_function:NN` ..... 2156  
`\seq_map_indexed_inline:Nn` .... 3882, 3896  
`\seq_map_inline:Nn` .....  
..... 2006, 2012, 2168, 2926, 4050, 4261,  
4263, 4265, 4387, 4389, 4391, 4658, 5225, 6021  
`\seq_mapthread_function:NNN` ..... 4959  
`\seq_new:N` ..... 267, 281,  
312, 313, 314, 315, 316, 317, 322, 356, 6298  
`\seq_put_right:Nn` ..... 4645  
`\seq_set_eq:NN` ..... 1373, 4019  
`\seq_set_filter:NNn` ..... 4021, 4048  
`\seq_set_from_clist:Nn` ..... 6299  
`\seq_set_map_x:Nnn` ..... 6304  
`\seq_use:Nnnn` ..... 2549, 4585, 6486, 6868  
`\l_tmpa_seq` ..... 4021, 4048  
`\l_tmpb_seq` ..... 4019, 4021, 4048, 4050  
`\setlist` ..... 372, 383, 680, 685, 696, 701  
siunitx commands:  
`\siunitx_cell_begin:w` ..... 176, 188  
`\siunitx_cell_end:` ..... 191  
skip commands:  
`\skip_gadd:Nn` ..... 2245  
`\skip_gset:Nn` ..... 2236  
`\skip_gset_eq:NN` ..... 2243, 2244  
`\skip_horizontal:N` .... 129, 146, 1305,  
1306, 1495, 1496, 1520, 1521, 1557, 1558,  
1561, 1562, 1580, 1581, 1611, 1617, 1704,  
1903, 1916, 1923, 2117, 2118, 2120, 2121,  
2192, 2193, 2205, 2207, 2223, 2225, 2247,  
2254, 2256, 2275, 2280, 2282, 2301, 2302,  
2360, 2361, 2362, 2365, 2400, 2405, 2406, 2407  
`\skip_horizontal:n` ..... 417, 1035, 1750  
`\skip_vertical:N` .....  
..... 133, 1081, 1083, 1553, 1564, 1997, 2022, 4677  
`\skip_vertical:n` ..... 931, 4507  
`\g_tmpa_skip` 2236, 2243, 2244, 2245, 2247, 2275  
`\c_zero_skip` ..... 1177

<code>\space</code> .....	289, 290
<code>\stepcounter</code> .....	403, 408
str commands:	
<code>\c_backslash_str</code> .....	289
<code>\c_colon_str</code> .....	797
<code>\str_case:nn</code> .....	
... 3617, 5404, 5416, 5438, 5450, 6066, 6075	
<code>\str_case:nnTF</code> .....	
... 1525, 1669, 1962, 4565, 6139, 6152	
<code>\str_clear_new:N</code> .....	5947
<code>\str_foldcase:n</code> .....	3617
<code>\str_gclear:N</code> .....	2629
<code>\str_gset:Nn</code> .....	
... 1454, 2418, 2427, 2456, 2467, 2475, 5684	
<code>\str_if_empty:NTF</code> .....	
... 898, 1000, 1090, 1253, 1271, 1453,	
2189, 2214, 2232, 2263, 2289, 2312, 2417,	
2426, 2496, 2517, 4951, 4978, 6087, 6101, 6111	
<code>\str_if_eq:nnTF</code> .....	
... 84, 288, 1070, 1696, 1699, 1743, 1820,	
1857, 1879, 1910, 2137, 2139, 2172, 5480, 5786	
<code>\str_if_eq_p:nn</code> .....	515, 1634,
... 1635, 1709, 4126, 4130, 4180, 4184, 5015, 5020	
<code>\str_if_in:NnTF</code> .....	1950, 2054
<code>\str_new:N</code> .....	284, 497, 796
<code>\str_range:Nnn</code> .....	1954, 2058
<code>\str_set:Nn</code> .....	641, 785, 5895
<code>\str_set_eq:NN</code> .....	645, 797
<code>\l_tmpa_str</code> .....	641, 642, 644, 645
<code>\strut</code> .....	2006, 2012
<code>\strutbox</code> .....	3109, 3127, 5989, 5992
<code>\SubMatrix</code> .....	1369, 2590,
... 5921, 6412, 6444, 6451, 6457, 6461, 6479, 6626	
sys commands:	
<code>\sys_if_engine_xetex_p:</code> .....	1032
<code>\sys_if_output_dvi_p:</code> .....	1032

## T

<code>\tabcolsep</code> .....	1063, 1557, 1561
<code>\tabskip</code> .....	1177
<code>\tabularnote</code> .....	367, 390, 397, 6551, 6571
<code>\tabularnotes</code> .....	2011
T <sub>E</sub> X and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:	
<code>\@BTnormal</code> .....	1152
<code>\@acol</code> .....	1053
<code>\@acoll</code> .....	1051
<code>\@acolr</code> .....	1052
<code>\@array@array</code> .....	1055
<code>\@arrayacol</code> .....	1051, 1052, 1053
<code>\@arstrutbox</code> .....	891, 892,
... 931, 1179, 1181, 1183, 1186, 1188, 1775, 1779	
<code>\@currentenv</code> .....	5786
<code>\@depth</code> .....	5838, 6177, 6201
<code>\@gobblethree</code> .....	240
<code>\@halignto</code> .....	1054, 1066, 1067
<code>\@height</code> ....	113, 128, 145, 5836, 6176, 6200
<code>\@ifclassloaded</code> .....	51, 54, 6272, 6282
<code>\@ifnextchar</code> .....	1225
<code>\@ifpackageloaded</code> .....	
... 30, 33, 36, 39, 75, 91, 167, 6275, 6285	
<code>\@mainaux</code> .....	76,
... 236, 2490, 2491, 2498, 2504, 2511, 2512,	
... 2519, 2525, 2530, 2531, 2533, 2551, 2618,	
... 2619, 2625, 4580, 4581, 4587, 4781, 4782, 4788	

<code>\@tabarray</code> .....	1068
<code>\@tempswafalse</code> .....	1605
<code>\@tempswattrue</code> .....	1604
<code>\@temptokena</code> .....	
... 182, 184, 202, 204, 220, 223, 1603, 1622	
<code>\@whiles</code> .....	1605
<code>\@width</code> .....	113, 5839, 6178, 6202
<code>\@xhline</code> .....	116
<code>\bBigg@</code> .....	1299, 1301
<code>\c@MaxMatrixCols</code> .....	2446, 6336
<code>\c@tabularnote</code> .....	1990, 2001, 2029
<code>\col@sep</code> .....	1063, 1064, 1520,
... 1581, 2192, 2245, 2301, 2365, 2400, 2996, 3018	
<code>\CT@arc</code> .....	95, 96
<code>\CT@arc@</code> .....	94, 99,
... 114, 127, 144, 160, 162, 1461, 2023, 2631,	
... 4356, 4479, 4742, 5481, 5539, 5586, 5748, 6020	
<code>\CT@dr</code> .....	102, 103
<code>\CT@drsc@</code> ..	101, 106, 4333, 4336, 4458, 4461
<code>\CT@everycr</code> .....	1170
<code>\CT@row@color</code> .....	1172
<code>\if@temp</code> .....	1605
<code>\NC@</code> .....	1128
<code>\NC@find</code> .....	193, 209, 221
<code>\NC@list</code> .....	1605
<code>\NC@rewrite@S</code> .....	180, 200, 222
<code>\new@ifnextchar</code> .....	1225
<code>\newcol@</code> .....	1130, 1131
<code>\nicematrix@redefine@check@rerun</code> ..	76, 79
<code>\pgf@relevantforpicturesizefalse</code> ....	
... 1333, 2667, 2694, 3319,	
... 3488, 3895, 4025, 4324, 4449, 4871, 4881,	
... 4892, 5323, 5477, 5538, 5585, 5735, 5796, 5951	
<code>\pgfsys@getposition</code> .....	
... 1331, 1338, 1346, 1426, 1429	
<code>\pgfsys@markposition</code> .....	
... 1017, 1025, 1082, 1330,	
... 2185, 2206, 2224, 2255, 2281, 2305, 2709, 2722	
<code>\pgfutil@check@rerun</code> .....	81, 82
<code>\reserved@a</code> .....	115
<code>\rvtx@ifformat@geq</code> .....	57
<code>\set@color</code> .....	5066, 5319
<code>\tikz@library@external@loaded</code> .....	1462
tex commands:	
<code>\tex_mkern:D</code> .....	63, 65, 67, 70
<code>\tex_the:D</code> .....	184, 204
<code>\textfont</code> .....	1976
<code>\textit</code> .....	359
<code>\textsuperscript</code> .....	360, 361
<code>\the</code> .....	153, 154, 1605, 1622
<code>\thetabularnote</code> .....	362
<code>\tikzexternaldisable</code> .....	1464
<code>\tikzset</code> .....	1355, 1466, 2593
tl commands:	
<code>\tl_clear:N</code> ....	4282, 4293, 4407, 4418, 5472
<code>\tl_clear_new:N</code> .....	3961, 3962, 4009,
... 4257, 4383, 5913, 5914, 5932, 5933, 5934, 5935	
<code>\tl_const:Nn</code> .....	
... 42, 43, 46, 47, 483, 2323, 2368, 6128	
<code>\tl_count:n</code> .....	1957, 2061
<code>\tl_gclear:N</code> .....	1607, 2605, 2612, 3900
<code>\tl_gclear_new:N</code> .....	
... 1228, 1229, 1230, 1231, 1232, 1233, 1234	



<b>3</b>	<b>The vertical position of the arrays</b>	<b>3</b>
<b>4</b>	<b>The blocks</b>	<b>4</b>
4.1	General case . . . . .	4
4.2	The mono-column blocks . . . . .	5
4.3	The mono-row blocks . . . . .	6
4.4	The mono-cell blocks . . . . .	6
4.5	A small remark . . . . .	6
<b>5</b>	<b>The rules</b>	<b>7</b>
5.1	Some differences with the classical environments . . . . .	7
5.1.1	The vertical rules . . . . .	7
5.1.2	The command <code>\cline</code> . . . . .	8
5.2	The thickness and the color of the rules . . . . .	8
5.3	The tools of <code>nicematrix</code> for the rules . . . . .	8
5.3.1	The keys <code>hlines</code> and <code>vlines</code> . . . . .	9
5.3.2	The key <code>hvlines</code> . . . . .	9
5.3.3	The (empty) corners . . . . .	9
5.4	The command <code>\diagbox</code> . . . . .	10
5.5	Dotted rules . . . . .	11
<b>6</b>	<b>The color of the rows and columns</b>	<b>11</b>
6.1	Use of <code>colortbl</code> . . . . .	11
6.2	The tools of <code>nicematrix</code> in the <code>\CodeBefore</code> . . . . .	12
6.3	Color tools with the syntax of <code>colortbl</code> . . . . .	15
<b>7</b>	<b>The width of the columns</b>	<b>16</b>
<b>8</b>	<b>The exterior rows and columns</b>	<b>17</b>
<b>9</b>	<b>The continuous dotted lines</b>	<b>18</b>
9.1	The option <code>nullify-dots</code> . . . . .	20
9.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code> . . . . .	20
9.3	How to generate the continuous dotted lines transparently . . . . .	21
9.4	The labels of the dotted lines . . . . .	22
9.5	Customisation of the dotted lines . . . . .	22
9.6	The dotted lines and the rules . . . . .	23
<b>10</b>	<b>The <code>\CodeAfter</code></b>	<b>24</b>
10.1	The command <code>\line</code> in the <code>\CodeAfter</code> . . . . .	24
10.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code> . . . . .	24
<b>11</b>	<b>The notes in the tabulars</b>	<b>26</b>
11.1	The footnotes . . . . .	26
11.2	The notes of tabular . . . . .	27
11.3	Customisation of the tabular notes . . . . .	28
11.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code> . . . . .	30
<b>12</b>	<b>Other features</b>	<b>30</b>
12.1	Use of the column type <code>S</code> of <code>siunitx</code> . . . . .	30
12.2	Alignment option in <code>{NiceMatrix}</code> . . . . .	31
12.3	The command <code>\rotate</code> . . . . .	31
12.4	The option <code>small</code> . . . . .	31
12.5	The counters <code>iRow</code> and <code>jCol</code> . . . . .	32
12.6	The option <code>light-syntax</code> . . . . .	33
12.7	Color of the delimiters . . . . .	33
12.8	The environment <code>{NiceArrayWithDelims}</code> . . . . .	33

<b>13</b>	<b>Use of Tikz with nicematrix</b>	<b>33</b>
13.1	The nodes corresponding to the contents of the cells . . . . .	33
13.2	The “medium nodes” and the “large nodes” . . . . .	34
13.3	The nodes which indicate the position of the rules . . . . .	36
13.4	The nodes corresponding to the command <code>\SubMatrix</code> . . . . .	37
<b>14</b>	<b>API for the developers</b>	<b>37</b>
<b>15</b>	<b>Technical remarks</b>	<b>38</b>
15.1	Definition of new column types . . . . .	38
15.2	Diagonal lines . . . . .	38
15.3	The “empty” cells . . . . .	39
15.4	The option <code>exterior-arraycolsep</code> . . . . .	39
15.5	Incompatibilities . . . . .	39
<b>16</b>	<b>Examples</b>	<b>40</b>
16.1	Notes in the tabulars . . . . .	40
16.2	Dotted lines . . . . .	41
16.3	Dotted lines which are no longer dotted . . . . .	42
16.4	Stacks of matrices . . . . .	43
16.5	How to highlight cells of a matrix . . . . .	45
16.6	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code> . . . . .	48
<b>17</b>	<b>Implementation</b>	<b>48</b>
<b>18</b>	<b>History</b>	<b>201</b>
	<b>Index</b>	<b>207</b>