

# The package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

February 25, 2023

## Abstract

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

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

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

The package `nicematrix` is entirely contained in the file `nicematrix.sty`. This file may be put in the current directory or in a `texmf` tree. However, the best is to install `nicematrix` with a TeX distribution such as MiKTeX, TeX Live 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 6.15 of `nicematrix`, at the date of 2023/02/25.

<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>{NiceTabularX}</code> | <code>{bNiceArray}</code> | <code>{bNiceMatrix}</code> |
|                             | <code>{BNiceArray}</code> | <code>{BNiceMatrix}</code> |
|                             | <code>{vNiceArray}</code> | <code>{vNiceMatrix}</code> |
|                             | <code>{VNiceArray}</code> | <code>{VNiceMatrix}</code> |

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

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

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

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

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

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

## 2 The vertical space between the rows

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

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

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

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

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

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

\begin{pNiceMatrix}
\frac12 & -\frac12 \\
\frac13 & \frac14
\end{pNiceMatrix}
```

---

<sup>3</sup>In fact, it's possible to use directly the `X` columns in the environment `{NiceTabular}` (and the required width for the tabular is fixed by the key `width`): cf. p. 23

<sup>4</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. 
$$\begin{array}{cccccc} n & 0 & 1 & 2 & 3 & 4 & 5 \\ u_n & 1 & 2 & 4 & 8 & 16 & 32 \end{array}$$

```

However, it's also possible to use the tools of `booktabs`<sup>5</sup>: `\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. 
$$\begin{array}{cccccc} n & 0 & 1 & 2 & 3 & 4 & 5 \\ u_n & 1 & 2 & 4 & 8 & 16 & 32 \end{array}$$

```

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

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

<sup>5</sup>The extension `booktabs` is *not* loaded by `nicematrix`.

## 4 The blocks

### 4.1 General case

In the environments of `nicematrix`, it's possible to use the command `\Block` in order to place an element in the center of a rectangle of merged cells of the array.<sup>6</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}`, `{NiceTabular*}` and `{NiceTabularX}`, the content of the block is composed in text mode whereas, in the other environments, it is composed in math mode.

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

```


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


```

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

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

```


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


```

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

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

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;
- the key `fill` takes in as value a color and fills the block with that color;
- the key `draw` takes in as value a color and strokes the frame of the block with that color (the default value of that key is the current color of the rules of the array);
- the key `color` takes in as value a color and apply that color the content of the block but draws also the frame of the block with that color;

<sup>6</sup>The spaces after a command `\Block` are deleted.

<sup>7</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 keys `hlines`, `vlines` and `hvlines` draw all the corresponding rules in the block;<sup>8</sup>
- the key `line-width` is the width of the rules (this key is meaningful only when one of the keys `draw`, `hvlines`, `vlines` and `hlines` is used);
- 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>9</sup>);
- when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz<sup>10</sup> by using as options the value of that key `tikz` (which must be a list of keys allowed for a Tikz path). For examples, cf. p. 51;
- the key `name` provides a name to the rectangular Tikz node corresponding to the block; it's possible to use that name with Tikz in the `\CodeAfter` of the environment (cf. p. 31);
- the key `respect-arraystretch` prevents the setting of `\arraystretch` to 1 at the beginning of the block (which is the behaviour by default) ;
- 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`; it's possible, in fact, in the list which is the value of the key `borders`, to add an entry of the form `tikz={list}` where `list` is a list of couples `key=value` of Tikz specifying the graphical characteristics of the lines that will be drawn (for an example, see p. 55).
- **Nouveau 6.15**  
By default, the rules are not drawn in the blocks (see the section about the rules: section 5 p. 9). However, if the key `transparent` is used, the rules are drawn. For an example, see section 18.1 on page 51.

There is also keys for the horizontal and vertical positions of the content of the block: cf. 4.5 p. 7).

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

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

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

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

<sup>8</sup>However, the rules are not drawn in the sub-blocks of the block, as always with `nicematrix`: the rules are not drawn in the blocks (cf. section 5 p. 9).

<sup>9</sup>This value is the initial value of the *rounded corners* of Tikz.

<sup>10</sup>Tikz should be loaded (by default, `nicematrix` only loads `PGF`) and, if it's not, an error will be raised.

## 4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

- The natural width of the contents of these blocks is taken into account for the width of the current column.

In the columns with a fixed width (columns `w{...}{...}`, `p{...}`, `b{...}`, `m{...}` and `X`), the content of the block is formatted as a paragraph of that width.

- The specification of the horizontal position provided by the type of column (`c`, `r` or `l`) is taken into account for the blocks (but the `\Block` may have its own specification of alignment: cf. 4.5 p. 7).
- 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).

|                |    |
|----------------|----|
| <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>11</sup>
- It's possible to draw one or several borders of the cell with the key `borders`.

---

<sup>11</sup>If one simply wishes to color the background of a unique cell, there is no point using the command `\Block`: it's possible to use the command `\cellcolor` (when the key `colortbl-like` is used).

```

\begin{NiceTabular}{cc}
\toprule
Writer & \Block[l]{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>12</sup>

## 4.5 Horizontal position of the content of the block

The command `\Block` accepts the keys `l`, `c` and `r` for the horizontal position of its content.

```

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

```

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

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

```

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

```

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

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

Here is the same example with the key `C` for the first block.

<sup>12</sup>One may consider that the default value of the first mandatory argument of `\Block` is `1-1`.

```

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

```

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

## 4.6 Vertical position of the content of the block

### New 6.14

For the vertical position, the commands `\Block` accepts the keys `v-center`<sup>13</sup>, `t`, `b`, `T` and `B`.

- With the key `v-center`, the content of the block is vertically centered.
- With the key `t`, the baseline of the content of the block is aligned With the baseline of the first row concerned by the block).
- with the key `b`, the baseline of the last row of the content of the block (we recall that the content of a block may contains several lines separated by `\\`) is aligned with the baseline of the last of the rows of the array involved in the block.
- With the key `T`, the content of the block is set upwards with only a margin equal to the PGF/Tikz parameter `inner ysep` (use `\pgfset` to change the value of that parameter).
- With the key `B`, the content of the block is set downwards with only a margin equal to the PGF/Tikz parameter `inner ysep`.

When no key is given, the key `v-center` applies (excepted in the mono-row blocks).

```
\NiceMatrixOptions{rules/color=[gray]{0.75}, hvlines}
```

```

\begin{NiceTabular}{ccc}
\Block[fill=red!10,t,l]{4-2}{two\\lines}
& & \Huge first \\
& & second \\
& & third \\
& & fourth \\
text & & text \\
\end{NiceTabular}

```

|              |              |  |
|--------------|--------------|--|
| two<br>lines | <b>first</b> |  |
|              | second       |  |
|              | third        |  |
|              | fourth       |  |
| text         | text         |  |

<sup>13</sup>That key could not have been named `c` since the key `c` is used for the horizontal alignment.



```

\begin{NiceTabular}{ccc}
\Block[fill=red!10,b,r]{4-2}{two\\lines}
& & \Huge first\\
& & second \\
& & third \\
& & fourth \\
text & text & \\
\end{NiceTabular}

```

|              |        |  |
|--------------|--------|--|
| two<br>lines | first  |  |
|              | second |  |
|              | third  |  |
|              | fourth |  |
| text         | text   |  |

```

\begin{NiceTabular}{ccc}
\Block[fill=red!10,T,l]{4-2}{two\\lines}
& & \Huge first\\
& & second \\
& & third \\
& & fourth \\
text & text & \\
\end{NiceTabular}

```

|              |        |  |
|--------------|--------|--|
| two<br>lines | first  |  |
|              | second |  |
|              | third  |  |
|              | fourth |  |
| text         | text   |  |

```

\begin{NiceTabular}{ccc}
\Block[fill=red!10,B,r]{4-2}{two\\lines}
& & \Huge first\\
& & second \\
& & third \\
& & fourth \\
text & text & \\
\end{NiceTabular}

```

|              |        |  |
|--------------|--------|--|
| two<br>lines | first  |  |
|              | second |  |
|              | third  |  |
|              | fourth |  |
| text         | text   |  |

## 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 the package `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. 12) nor in the potential exterior rows (created by the keys `first-row` and `last-row`: cf. p. 24).

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}}

```

### 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 | B | C | D |

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

```

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

```

|   |   |   |   |
|---|---|---|---|
| A | B | C | D |
| A | B | C | D |

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

## 5.2 The thickness and the color of the rules

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

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

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

```

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

```

|        |        |          |
|--------|--------|----------|
| rose   | tulipe | lys      |
| arum   | iris   | violette |
| muguet | dahlia | souci    |

### 5.3 The tools of nicematrix for the rules

Here are the tools provided by nicematrix for the rules.

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

All these tools don’t draw the rules in the blocks nor in the empty corners (when the key `corners` is used), nor in the exterior rows and columns.

- These blocks are:
  - the blocks created by the command `\Block`<sup>14</sup> presented p. 4;
  - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `\Vdots`, etc. (cf. p. 25).
- The corners are created by the key `corners` explained below (see p. 12).
- For the exterior rows and columns, see p. 24.

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

The key `\Hline` takes in an optional argument (between square brackets) which is a list of *key=value* pairs. For the description of those keys, see `custom-line` on p. 13.

#### 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.<sup>15</sup>

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}$
```

$$\left( \begin{array}{c|c|c|c|c|c} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 2 & 3 & 4 & 5 & 6 \end{array} \right)$$

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

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

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

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

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

<sup>14</sup>And also the command `\multicolumn` but it’s recommended to use instead `\Block` in the environments of nicematrix.

<sup>15</sup>It’s possible to put in that list some intervals of integers with the syntax *i-j*.

### 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>16</sup>

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

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

|   |   |   |   |   |   |
|---|---|---|---|---|---|
|   |   |   |   | A |   |
|   |   | A | A | A |   |
|   |   |   | A |   |   |
|   |   | A | A | A | A |
| A | A | A | A | A | A |
| A | A | A | A | A | A |
|   | A | A | A |   |   |
|   |   | B | A |   |   |
|   |   |   | A |   |   |

When the key `corners` is used, `nicematrix` computes the (empty) corners and these corners will be taken into account by the tools for drawing the rules (the rules won't be drawn in the corners).

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

|   |   |   |   |   |   |
|---|---|---|---|---|---|
|   |   |   |   |   | A |
|   |   | A | A | A |   |
|   |   |   | A |   |   |
|   |   | A | A | A | A |
| A | A | A | A | A | A |
| A | A | A | A | A | A |
|   | A | A | A |   |   |
|   |   | B |   | A |   |
|   |   |   |   | A |   |

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

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

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 1 |   |   |   |   |   |
| 1 | 1 |   |   |   |   |
| 1 | 2 | 1 |   |   |   |
| 1 | 3 | 3 | 1 |   |   |
| 1 | 4 | 6 | 4 | 1 |   |
|   |   |   |   |   | 1 |

▷ The corners are also taken into account by the tools provided by `nicematrix` to color cells, rows and columns. These tools don't color the cells which are in the corners (cf. p. 16).

<sup>16</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. The precise definition of a "non-empty cell" is given below (cf. p. 49).

## 5.4 The command `\diagbox`

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

```

\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}

```

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

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

## 5.5 Commands for customized rules

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

- the key `command` is the name (without the backslash) of a command that will be created by `nicematrix` and that will be available for the final user in order to draw horizontal rules (similarly to `\hline`);
- the key `ccommand` is the name (without the backslash) of a command that will be created by `nicematrix` and that will be available for the final user to order to draw partial horizontal rules (similarly to `\cline`, hence the name `ccommand`): the argument of that command is a list of intervals of columns specified by the syntax  $i$  or  $i-j$ .<sup>17</sup>
- the key `letter` takes in as argument a letter<sup>18</sup> that the user will use in the preamble of an environment with preamble (such as `\NiceTabular`) in order to specify a vertical rule.

We will now speak of the keys which describe the rule itself. Those keys may also be used in the (optional) argument of an individual command `\Hline`.

There is three possibilities.

- *First possibility*

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

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

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

<sup>17</sup>It's recommended to use such commands only once in a row because each use will create space between the rows corresponding to the total width of the rule.

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

```

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

```

- *Second possibility*

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

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

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

Here is an example with the key `dotted` of Tikz.

```

\NiceMatrixOptions
{
  custom-line =
  {
    letter = I ,
    tikz = dotted ,
    total-width = \pgflinewidth
  }
}

```

|       |       |       |
|-------|-------|-------|
| one   | two   | three |
| four  | five  | six   |
| seven | eight | nine  |

```

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

```

- *Third possibility* : the key `dotted`

As one can see, the dots of a dotted line of Tikz have the shape of a square, and not a circle. That's why the extension `nicematrix` provides in the key `custom-line` a key `dotted` which will draw rounded dots. The initial value of the key `total-width` is, in this case, equal to the diameter of the dots (but the user may change the value with the key `total-width` if needed). Those dotted rules are also used by `nicematrix` to draw continuous dotted rules between cells of the matrix with `\Cdots`, `\Vdots`, etc. (cf. p. 25).

In fact, `nicematrix` defines by default the commands `\hdottedline` and `\cdottedline` and the letter “:” for those dotted rules.<sup>19</sup>

<sup>19</sup>However, it's possible to overwrite those definitions with a `custom-line` (in order, for example, to switch to dashed lines).

```

\NiceMatrixOptions % present in nicematrix.sty
{
  custom-line =
  {
    letter = : ,
    command = hdottedline ,
    ccommand = cdottedline ,
    dotted
  }
}

```

Thus, it's possible to use the commands `\hdottedline` and `\cdottedline` to draw horizontal dotted rules.

```

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

```

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

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)

```

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

## 6 The color of the rows and columns

### 6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

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

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

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

The package `nicematrix` provides some tools (independent of `colortbl`) to draw the colored panels first, and, then, the content of the cells and the rules. This strategy is more conform to the “painting model” of the formats PostScript and PDF and is more suitable for the PDF viewers. However, it requires several compilations.<sup>20</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 environment
\end{pNiceArray}
```

New commands are available in that `\CodeBefore`: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors`, `\rowlistcolors`, `\chessboardcolors` and `\arraycolor`.<sup>21</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.

These commands don’t color the cells which are in the “corners” if the key `corners` is used. This key has been described p. 12.

- 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. In fact, despite its name, this command may be used to color a whole row (with the syntax  $i-$ ) or a whole column (with the syntax  $-j$ ).

```
\begin{NiceTabular}{ccc}[hvlines]
\CodeBefore
  \cellcolor[HTML]{FFFF88}{3-1,2-2,-3}
\Body
a & b & c \\
e & f & g \\
h & i & j \\
\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}{ccc}[hvlines]
\CodeBefore
  \rectanglecolor{blue!15}{2-2}{3-3}
\Body
a & b & c \\
e & f & g \\
h & i & j \\
\end{NiceTabular}
```

|   |   |   |
|---|---|---|
| a | b | c |
| e | f | g |
| h | i | j |

<sup>20</sup>If you use Overleaf, Overleaf will do automatically the right number of compilations.

<sup>21</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. 46.



- 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. 24). 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. 40).

- 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).

```

 $\begin{NiceArray}{111}[hvlines]
\CodeBefore
  \rowcolor{red!15}{1,3-5,8-}
\Body
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3 \\
a_4 & b_4 & c_4 \\
a_5 & b_5 & c_5 \\
a_6 & b_6 & c_6 \\
a_7 & b_7 & c_7 \\
a_8 & b_8 & c_8 \\
a_9 & b_9 & c_9 \\
a_{10} & b_{10} & c_{10}
\end{NiceArray}$ 

```

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

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

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

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

- The key `cols` describes a set of columns. The command `\rowcolors` will color only the cells of these columns. The value is a comma-separated list of intervals of the form  $i-j$  (where  $i$  or  $j$  may be replaced by  $*$ ).

- With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.<sup>22</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}{lr}[hvlines]
\CodeBefore
  \rowcolors{1}{blue!10}{}[respect-blocks]
\Body
\Block{2-1}{John} & 12 \
& 13 \
Steph & 8 \
\Block{3-1}{Sarah} & 18 \
& 17 \
& 15 \
Ashley & 20 \
Henry & 14 \
\Block{2-1}{Madison} & 15 \
& 19
\end{NiceTabular}
```

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

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

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

|           |
|-----------|
| Peter     |
| James     |
| Abigail   |
| Elisabeth |
| Claudius  |
| Jane      |
| Alexandra |

It's also possible to use in the command `\rowlistcolors` a color series defined by the command `\definecolorseries` of `xcolor` (and initialized with the command `\resetcolorseries`<sup>23</sup>).

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

<sup>23</sup>For the initialization, in the following example, you have used the counter `iRow` which, when used in the `\CodeBefore`

```

\begin{NiceTabular}{c}
\CodeBefore
  \definecolorseries{BlueWhite}{rgb}{last}{blue}{white}
  \resetcolorseries{\value{iRow}}{BlueWhite}
  \rowlistcolors{1}{BlueWhite!+}
\Body
Peter \\
James \\
Abigail \\
Elisabeth \\
Claudius \\
Jane \\
Alexandra \\
\end{NiceTabular}

```

|           |
|-----------|
| Peter     |
| James     |
| Abigail   |
| Elisabeth |
| Claudius  |
| Jane      |
| Alexandra |

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

```

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

```

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

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

```

\begin{NiceTabular}[c]{lSSSS}
\CodeBefore
  \rowcolor{red!15}{1-2}
  \rowcolors{3}{blue!15}{}
\Body
\toprule
\Block{2-1}{Product} &
\Block{1-3}{dimensions (cm)} & & &
\Block{2-1}{\rotate Price} \\
\cmidrule{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    |

---

(and in the `\CodeAfter`) corresponds to the number of rows of the array: cf. p 41. That leads to an ajustement of the gradation of the colors to the size of the tabular.

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>24</sup>

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

- `\cellcolor` which colorizes a cell;<sup>25</sup>
- `\rowcolor` which must be used in a cell and which colorizes the end of the row;
- `\columncolor` which must be used in the preamble of the environment with the same syntax as the corresponding command of `colortbl` (however, unlike the command `\columncolor` of `colortbl`, this command `\columncolor` can appear within another command, itself used in the preamble of the array).

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

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

## 7 The command `\RowStyle`

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

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

- The key `nb-rows` sets the number of rows to which the specifications of the current command will apply (with the special value `*`, it will apply to all the following rows).
- The keys `cell-space-top-limit`, `cell-space-bottom-limit` and `cell-space-limits` are available with the same meaning that the corresponding global keys (cf. p. 2).

<sup>24</sup>Up to now, this key is *not* available in `\NiceMatrixOptions`.

<sup>25</sup>However, this command `\cellcolor` will delete the following spaces, which does not the command `\cellcolor` of `colortbl`.

- The key `rowcolor` sets the color of the background and the key `color` sets the color of the text.<sup>26</sup>
- The key `bold` enforces bold characters for the cells of the row, both in math and text mode.

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

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

The command `\rotate` is described p. 40.

## 8 The width of the columns

### 8.1 Basic tools

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

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

|        |          |        |
|--------|----------|--------|
| Paris  | New York | Madrid |
| Berlin | London   | Roma   |
| Rio    | Tokyo    | Oslo   |

In the environments of `nicematrix`, it's also possible to fix the *minimal* width of all the columns (excepted the potential exterior columns: cf. p. 24) 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>27</sup>

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

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

<sup>26</sup>The key `color` uses the command `\color` but inserts also an instruction `\leavevmode` before. This instruction prevents a extra vertical space in the cells which belong to columns of type `p`, `b`, `m` and `X` (which start in vertical mode).

<sup>27</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).

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

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

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

But it's also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment `{NiceMatrixBlock}` with the option `auto-columns-width`<sup>28</sup>. The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

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

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

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

## 8.2 The columns V of varwidth

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

That point is illustrated on the following examples.

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

- first item
  - second item

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

- first item
  - second item

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

When the package `varwidth` is loaded, the columns `V` of `varwidth` are supported by `nicematrix`.

---

<sup>28</sup>At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

```

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

```

|                               |           |                               |
|-------------------------------|-----------|-------------------------------|
|                               | some text | some very very very long text |
| some very very very long text |           |                               |
| some very very very long text |           |                               |

Concerning `nicematrix`, one of the interests of this type of columns is that, for a cell of a column of type `V`, the PGF/Tikz node created by `nicematrix` for the content of that cell has a width adjusted to the content of the cell : cf. p. 44.

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

### 8.3 The columns X

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

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

For sake of similarity with the environment `{tabularx}`, `nicematrix` also provides an environment `{NiceTabularX}` with a syntax similar to the syntax of `{tabularx}`, that is to say with a first mandatory argument which is the width of the tabular.<sup>29</sup>

As with the packages `tabu`<sup>30</sup> and `tabularray`, the specifier `X` takes in an optional argument (between square brackets) which is a list of keys.

- It's possible to give a weight for the column by providing a positive integer directly as argument of the specifier `X`. For example, a column `X[2]` will have a width double of the width of a column `X` (which has a weight equal to 1).<sup>31</sup>
- It's possible to specify an horizontal alignment with one of the letters `l`, `c` and `r` (which insert respectively `\raggedright`, `\centering` and `\raggedleft` followed by `\arraybackslash`).
- It's possible to specify a vertical alignment with one of the keys `t` (alias `p`), `m` and `b` (which construct respectively columns of type `p`, `m` and `b`). The default value is `t`.

```

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

```

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

<sup>29</sup>If `tabularx` is loaded, one must use `{NiceTabularX}` (and not `{NiceTabular}`) in order to use the columns `X` (this point comes from a conflict in the definitions of the specifier `X`).

<sup>30</sup>The extension `tabu` is now considered as deprecated.

<sup>31</sup>The negative values of the weight, as provided by `tabu` (which is now obsolete), are *not* supported by `nicematrix`. If such a value is used, an error will be raised.

## 9 The exterior rows and columns

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

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

```

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

```

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

The dotted lines have been drawn with the tools presented p. 25.

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>32</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. 42) 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 and the number of columns.
  - In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.  
*However, it's possible to provide the number of the last row and the number of the last column as values of the options `last-row` and `last-col`, tending to an acceleration of the whole compilation of the document. That's what we will do throughout the rest of the document.*

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

---

<sup>32</sup>The users wishing exterior columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 32).



```

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

```

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

### Remarks

- As shown in the previous example, the horizontal and vertical rules don't extend in the exterior rows and columns. This remark also applies to the customized rules created by the key `custom-line` (cf. p. 13).
- A specification of color present in `code-for-first-row` also applies to a dotted line drawn in that exterior “first row” (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 21) 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. 32.

## 10 The continuous dotted lines

Inside the environments of the package `nicematrix`, new commands are defined: `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots`. These commands are intended to be used in place of `\dots`, `\cdots`, `\vdots`, `\ddots` and `\iddots`.<sup>33</sup>

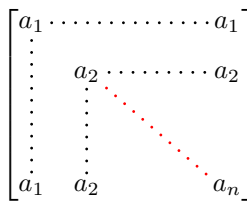
Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells<sup>34</sup> on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it's an horizontal line; for `\Vdots`, it's a vertical line and for `\Ddots` and `\Iddots` diagonal ones. It's possible

<sup>33</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>34</sup>The precise definition of a “non-empty cell” is given below (cf. p. 49).

to change the color of these lines with the option `color`.<sup>35</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}
```



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}
```



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}
```



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

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

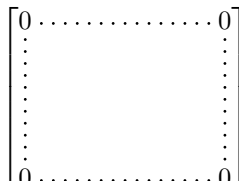
```
\begin{bNiceMatrix}
0        & \Cdots &      & 0      & \\
\Vdots   &        &      & \Vdots & \\
          &        &      & \Vdots & \\
0        &        & \Cdots & 0      & \\
\end{bNiceMatrix}
```



There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\Vdots` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.<sup>36</sup>

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}
```



<sup>35</sup>It's also possible to change the color of all these dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 29.

<sup>36</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. 21

## 10.1 The option nullify-dots

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`<sup>37</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}
```

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

### 10.3 How to generate the continuous dotted lines transparently

Imagine you have a document with a great number of mathematical matrices with ellipsis. You may wish to use the dotted lines of `nicematrix` without having to modify the code of each matrix. It's possible with the keys `renew-dots` and `renew-matrix`.<sup>38</sup>

- The option `renew-dots`  
 With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`<sup>33</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.

<sup>37</sup>We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

<sup>38</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`.

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}
```

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

## 10.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `\CodeAfter` which is described p. 31) accept two optional arguments specified by the tokens `_` and `^` for labels positioned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

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

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

## 10.5 Customisation of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and `\Vdotsfor` (and by the command `\line` in the `\CodeAfter` which is described p. 31) may be customized by the following options (specified between square brackets after the command):

- `color`;
- `radius`;
- `shorten-start`, `shorten-end` and `shorten`;
- `inter`;
- `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` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.), and, thus have for names:

- `xdots/color`;
- `xdots/radius`;
- `xdots/shorten-start`, `xdots/shorten-end` and `xdots/shorten`;
- `xdots/inter`;
- `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. 24.

### The option `xdots/radius`

The option `radius` fixes the radius of the dots. The initial value is 0.53 pt.

### The option `xdots/shorten`

The keys `xdots/shorten-start` and `xdots/shorten-end` fix the margin at the extremities of the line. The key `xdots/shorten` fixes both parameters. The initial value is 0.3 em (it is recommended to use a unit of length dependent of the current font).

### The option `xdots/inter`

The option `xdots/inter` fixes the length between the dots. The initial value is 0.45 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>39</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 paths (with the exception of “color”, “shorten >” and “shorten <”).

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

```


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


```

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

## 10.6 The dotted lines and the rules

The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier `|` in the preamble, by the command `\Hline`, by the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` and by the tools created by `custom-line` are not drawn within the blocks).<sup>40</sup>

```


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


```

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

<sup>39</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. Nevertheless, you can have a look at the following page to see how to have dotted rules with rounded dots in Tikz: <https://tex.stackexchange.com/questions/52848/tikz-line-with-large-dots>

<sup>40</sup>On the other side, the command `\line` in the `\CodeAfter` (cf. p. 31) does *not* create block.

## 11 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.<sup>41</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).<sup>42</sup>

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. 43.

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

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

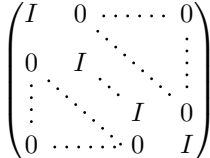
The command `\line` draws directly dotted lines between cells or blocks. It takes in two arguments for the cells or blocks to link. Both argument may be:

- a specification of cell of the form  $i$ - $j$  where  $i$  is the number of the row and  $j$  is the number of the column;
- the name of a block (created by the command `\Block` with the key `name` of that command).

The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 29).

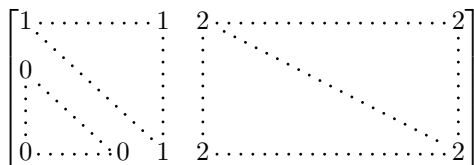
This command may be used, for example, to draw a dotted line between two adjacent cells.

```
\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}
```



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. 49).

```
\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}
```



<sup>41</sup>There is also a key `code-before` described p. 16.

<sup>42</sup>Here are the keys accepted in that argument: `delimiters/color`, `rules` and its sub-keys and `sub-matrix` (linked to the command `\SubMatrix`) and its sub-keys.

Despite its name, the command `\SubMatrix` may also be used within a `{NiceTabular}`. Here is an example (which uses `\bottomrule` and `\toprule` of `booktabs`).

```

\begin{NiceTabular}{{}ll{}}
\toprule
Part A          & the first part \\
\Block{2-1}{Part B} & a first sub-part \\
                  & a second sub-part \\
\bottomrule
\CodeAfter
  \SubMatrix{\}{2-2}{3-2}{.}
\end{NiceTabular}

```

$$\begin{array}{l}
 \hline
 \text{Part A} \quad \text{the first part} \\
 \text{Part B} \left\{ \begin{array}{l} \text{a first sub-part} \\ \text{a second sub-part} \end{array} \right. \\
 \hline
 \end{array}$$

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

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

- the first argument is the left delimiter, which may be any extensible delimiter provided by LaTeX : `(`, `[`, `\{`, `\langle`, `\lgroup`, `\lfloor`, etc. but also the null delimiter `.`;
- the second argument is the upper-left corner of the submatrix with the syntax  $i$ - $j$  where  $i$  the number of row and  $j$  the number of column;
- the third argument is the lower-right corner with the same syntax;
- the fourth argument is the right delimiter;
- the last argument, which is optional, is a list of *key=value* pairs.<sup>43</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.

```

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

```

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

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

```

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

```

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

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

- `left-xshift` and `right-xshift` shift horizontally the delimiters (there exists also the key `xshift` which fixes both parameters);

<sup>43</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}]`).



- `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 these 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.

```


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


```

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

```


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


```

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. 47.

It's also possible to specify some delimiters<sup>44</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.

---

<sup>44</sup>Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

```


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


```

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

The command `\SubMatrix` is, in fact, also available in the `\CodeBefore`. By using `\SubMatrix` in the `\CodeBefore`, the delimiters drawn by those commands `\SubMatrix` are taken into account to limit the continuous dotted lines (drawn by `\Cdots`, `\Vdots`, etc.) which have an open extremity. For an example, see voir [18.8 p. 61](#).

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

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

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

```

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

```

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

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

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

```

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

```

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

## 12 Captions and notes in the tabulars

### 12.1 Caption of a tabular

The environment `{NiceTabular}` provides the keys `caption`, `short-caption` and `label` which may be used when the tabular is inserted in a floating environment (typically the environment `{table}`). With the key `caption`, the caption, when it is long, is wrapped at the width of the tabular (excepted the potential exterior columns specified by `first-col` and `last-col`), without the use of the package `threeparttable` or the package `floatrow`.

By default, the caption is composed below the tabular. With the key `caption-above`, available in `\NiceMatrixOptions`, the caption will be composed above de tabular.

The key `short-caption` corresponds to the optional argument of the classical command `\caption` and the key `label` corresponds, of course, to the command `\label`.

See table 1, p. 37 for an example of use the keys `caption` and `label`.

### 12.2 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) ant it is used to extract footnotes.

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

### 12.3 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 specified by `first-col` and `last-col`). With no surprise, that command is available only in the environments `{NiceTabular}`, `{NiceTabular*}` and `{NiceTabularX}`.

In fact, this command is available only if the extension `enumitem` has been loaded (before or after `nicematrix`). Indeed, the notes are composed at the end of the array with a type of list provided by the package `enumitem`.

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

| Last name             | First name | Birth day        |
|-----------------------|------------|------------------|
| Achard <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.

An alternative syntaxe is available with the environment `{TabularNote}`. That environment should be used at the end of the environment `{NiceTabular}` (but *before* a potential instruction `\CodeAfter`).

- 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 (or in a command `\captionof` of the package `caption`). It's also possible, as expected, to use the command `\tabularnote` in the caption provided by the *key* `caption` of the environment `{NiceTabular}`.

If several commands `\tabularnote` are used in a tabular with the same argument, only one note is inserted at the end of the tabular (but all the labels are composed, of course). It's possible to control that feature with the key `notes/detect-duplicates`.<sup>45</sup>

- 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. 37. This table has been composed with the following code (the package `caption` has been loaded in this document).

```

\begin{table}
\centering
\NiceMatrixOptions{caption-above}
\begin{NiceTabular}{@{}llc@{}}
[
caption = A tabular whose caption has been specified by the key
\texttt{caption}\tabularnote{It's possible to put a tabular note in the caption} ,
label = t:tabularnote ,
tabularnote = Some text before the notes. ,
notes/bottomrule
]

```

<sup>45</sup>For technical reasons, the final user is not allowed to put several commands `\tabularnote` with exactly the same argument in the caption of the tabular.

```

\toprule
Last name & First name & Length of life \\
\midrule
Churchill & Wiston & 91\\
Nightingale\table{Considered as the first nurse of history}
\table{Nicknamed ``the Lady with the Lamp''.}
& Florence\table{This note is shared by two references.} & 90 \\
Schoelcher & Victor & 89\table{The label of the note is overlapping.}\\
Touchet & Marie\table{This note is shared by two references.} & 89 \\
Wallis & John & 87 \\
\bottomrule
\end{NiceTabular}
\end{table}

```

Table 1: A tabular whose caption has been specified by the key `captiona`

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

Some text before the notes.

<sup>a</sup> It's possible to put a tabular 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> This note is shared by two references.

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

## 12.4 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`. This style of list is defined as follows (with, of course, keys of `enumitem`):

```
noitemsep , leftmargin = * , align = left , labelsep = Opt
```

The specification `align = left` in that style requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 37).

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

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

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

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

Initial value: `empty`

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

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

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

- The key `notes/detect-duplicates` activates the detection of the commands `\tabularnotes` with the same argument.

Initial value : `true`

For an example of customisation of the tabular notes, see p. 52.

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

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

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

## 13 Other features

## 14 Autres fonctionnalités

### 14.1 Command `\ShowCellNames`

The command `\ShowCellNames`, which may be used in the `\CodeBefore` and in the `\CodeAfter` display the name (with the form  $i-j$ ) of each cell. When used in the `\CodeAfter`, that command

applies a semi-transparent white rectangle to fade the array (caution: some PDF readers don't support transparency).

```
\begin{NiceTabular}{ccc}[hvlines,cell-space-limits=3pt]
  \Block{2-2}{} & & & test \\
  & & & blabla \\
  & & & some text & nothing \\
\CodeAfter \ShowCellNames
\end{NiceTabular}
```

|     |     |     |
|-----|-----|-----|
| 1-1 | 1-2 | 1-3 |
| 2-1 | 2-2 | 2-3 |
| 3-1 | 3-2 | 3-3 |

## 14.2 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}{ScW{c}{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}

```

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

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

## 14.3 Default column type in {NiceMatrix}

The environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.) and the commande `\pAutoNiceMatrix` (and its variants) provide an option `columns-type` to specify the type of column which will be used (the initial value is, of course, `c`).

The keys `l` and `r` are shortcuts for `columns-type=l` and `columns-type=r`.

```

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

```

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

The key `columns-type` is available in `\NiceMatrixOptions` but with the prefix `matrix`, which means that its name is, within `\NiceMatrixOptions` : `matrix/columns-type`.

## 14.4 The command \rotate

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

In the following command, we use that command in the `code-for-first-row`.<sup>46</sup>

```

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

```

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

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

<sup>46</sup>It can also be used in `\RowStyle` (cf. p. 20).



```

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

```

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

image of  $e_1$   $e_2$   $e_3$

## 14.5 The option `small`

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

```

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

```

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

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

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

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

## 14.6 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>47</sup>. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 16) and in the `\CodeAfter` (cf. p. 31), `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).

---

<sup>47</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.

```

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

```

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

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

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

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

```

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

```

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

## 14.7 The key light-syntax

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

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

```

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

```

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

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

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.<sup>48</sup>

## 14.8 Color of the delimiters

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

```

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

```

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

This colour also applies to the delimiters drawn by the command `\SubMatrix` (cf. p. 32).

<sup>48</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.

## 14.9 The environment `{NiceArrayWithDelims}`

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

```

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

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

## 14.10 The command `\OnlyMainNiceMatrix`

The command `\OnlyMainNiceMatrix` executes its argument only when it is in the main part of the array, that is to say it is not in one of the exterior rows. If it is used outside an environment of `nicematrix`, that command is no-op.

For an example of utilisation, see [tex.stackexchange.com/questions/488566](https://tex.stackexchange.com/questions/488566)

# 15 Use of Tikz with `nicematrix`

## 15.1 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>49</sup>

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`.

---

<sup>49</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. 25) and the computation of the “corners” (cf. p. 12).

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

```

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

```

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

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 59).

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

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

### 15.1.1 The columns V of varwidth

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

```

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

```

**Titre**

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

We have used the nodes corresponding to the position of the potential rules, which are described below (cf. p. 46).

## 15.2 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>50</sup>

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

---

<sup>50</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 “medium nodes” are constructed by adding the suffix “-medium” to the names of the “normal nodes”. In the following example, we have underlined the “medium nodes”. We consider that this example is self-explanatory.

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

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.<sup>51</sup>

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

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.<sup>52</sup>

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

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

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

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

Here is an array composed with the following code:

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

<sup>51</sup>There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 24).

<sup>52</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.

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

|        |          |         |
|--------|----------|---------|
| fraise | amande   | abricot |
| prune  | pêche    | poire   |
| noix   | noisette | brugnon |

The nodes we have described are not available by default in the `\CodeBefore` (described p. 16). It’s possible to have these nodes available in the `\CodeBefore` by using the key `create-cell-nodes` of the keyword `\CodeBefore` (in that case, the nodes are created first before the construction of the array by using informations written on the `aux` file and created a second time during the construction of the array itself).

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

```

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

```

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

### 15.3 The nodes which indicate the position of the rules

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

|        |     |     |          |       |
|--------|-----|-----|----------|-------|
|        | 1.5 | 2   | tulipe   | lys   |
| arum   |     | 2.5 | violette | mauve |
| muguet |     | 3   |          | 3.5   |

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

```

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

```

```

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

```

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

```

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

```

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

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

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 32.

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}\}`).

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

## 16 API for the developers

The package `nicematrix` provides two variables which are internal but public<sup>53</sup>:

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

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

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

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

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

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

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

Here is an example of utilisation:

```
\begin{NiceTabular}{ccc}[hvlines]
\CodeBefore
  \arraycolor{gray!10}
\Body
merlan & requin & cabillaud \\
baleine & \crossbox[red] & morue \\
mante & raie & poule
\end{NiceTabular}
```

|         |              |           |
|---------|--------------|-----------|
| merlan  | requin       | cabillaud |
| baleine | <del> </del> | morue     |
| mante   | raie         | poule     |

<sup>53</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.



## 17 Technical remarks

First remark: the package `underscore` must be loaded before `nicematrix`.

### 17.1 Diagonal lines

By default, all the diagonal lines<sup>54</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):

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

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

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

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

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

The same example without parallelization:

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

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]`.

### 17.2 The “empty” cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides. When the key `corners` is used (cf. p. 12), `nicematrix` computes corners consisting of empty cells. However, an “empty cell” is not necessarily a cell with no TeX content (that is to say a cell with no token between the two ampersands `&`). The precise rules are as follow.

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

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

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

<sup>54</sup>We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in the `\CodeAfter`.

- For the columns of type `p`, `m`, `b`, `V`<sup>55</sup> and `X`<sup>56</sup>, the cell is empty if (and only if) its content in the TeX code is empty (there is only spaces between the ampersands `&`).
- For the columns of type `c`, `l`, `r` and `w{\dots}{\dots}`, the cell is empty if (and only if) its TeX output has a width equal to zero.
- A cell containing the command `\NotEmpty` is not empty (and a PGF/Tikz node is created in that cell).
- A cell with only 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`.

### 17.3 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>57</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>58</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}` of `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).

### 17.4 Incompatibilities

The package `nicematrix` is not compatible with the class `ieeaccess` (because that class is not compatible with PGF/Tikz).<sup>59</sup>

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

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

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

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

The package `nicematrix` is not fully compatible with the packages and classes of `LuaTeX-ja`: the detection of the empty corners (cf. p. 12) may be wrong in some circumstances.

<sup>55</sup>The columns of type `V` are provided by `varwidth`: cf. p. 22.

<sup>56</sup>See p. 23

<sup>57</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>58</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.

<sup>59</sup>See <https://tex.stackexchange.com/questions/528975/error-loading-tikz-in-ieeeaccess-class>

The package `nicematrix` is not fully compatible with the package `arydshln` (because this package redefines many internals of `array`). By any means, in the context of `nicematrix`, it's recommended to draw dashed rules with the tools provided by `nicematrix`, by creating a customized line style with `custom-line`: cf. p. 13.

The columns `d` of `dcolumn` are not supported (but it's possible to use the columns `S` of `siunitx`).

## 18 Examples

### 18.1 Utilisation of the key “tikz” of the command `\Block`

The key `tikz` of the command `\Block` is available only when Tikz is loaded.<sup>60</sup> For the following example, we also need the Tikz library `patterns`.

```
\usetikzlibrary{patterns}

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

|  |   |  |
|--|---|--|
| pattern = grid,<br>pattern color = lightgray           | pattern = north west lines,<br>pattern color = blue | outer color = red!50,<br>inner color = white |
| pattern = sixpointed stars,<br>pattern color = blue!15 | left color = blue!50                                |  |

In the following example, we use the key `tikz` to hatch a row of the tabular. Remark that you use the key `transparent` of the command `\Block` in order to have the rules drawn in the block.<sup>61</sup>

```
\begin{NiceTabular}{ccc}[hvlines]
\CodeBefore
  \columncolor[RGB]{169,208,142}{2}
\Body
one & two & three \ \
\Block[transparent, tikz={pattern = north west lines, pattern color = gray}]{1-*}{}
four & five & six \ \
seven & eight & nine
```

|       |       |       |
|-------|-------|-------|
| one   | two   | three |
| four  | five  | six   |
| seven | eight | nine  |

<sup>60</sup>By default, `nicematrix` only loads PGF, which is a sub-layer of Tikz.

<sup>61</sup>By default, the rules are not drawn in the blocks created by the command `\Block`: cf. section 5 p. 9

## 18.2 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 12 p. 35.

Let's consider that we wish to number the notes of a tabular with stars.<sup>62</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>63</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{{}}
\toprule \RowStyle{\bfseries}
Last name & First name & Birth day \\
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.}
& Jacques & 5 juin 1962 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.}
& Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}
```

---

<sup>62</sup>Of course, it's realistic only when there is very few notes in the tabular.

<sup>63</sup>In fact: the value of its argument.

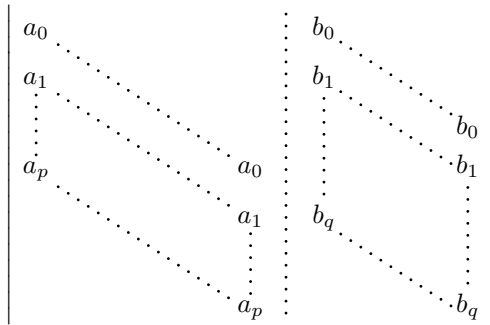
| Last name  | First name | Birth day        |
|------------|------------|------------------|
| Achard*    | Jacques    | June 5, 2005     |
| Lefebvre** | Mathilde   | January 23, 1975 |
| Vanesse    | Stephany   | October 30, 1994 |
| Dupont     | Chantal    | January 15, 1998 |

\*Achard is an old family of the Poitou.  
 \*\*The name Lefebvre is an alteration of the name Lefebure.

### 18.3 Dotted lines

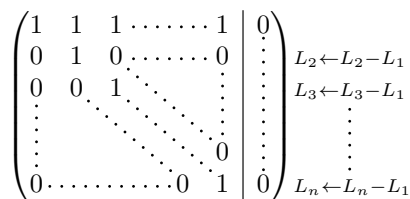
An example with the resultant of two polynoms:

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



An example for a linear system:

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



## 18.4 Dotted lines which are no longer dotted

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

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

In fact, it's even possible to draw solid lines with the commands `\Cdots`, `\Vdots`, etc.<sup>64</sup>

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

<sup>64</sup>In this document, the Tikz library `arrows.meta` has been loaded, which impacts the shape of the arrow tips.

## 18.5 Dashed rules

In the following example, we use the command `\Block` to draw dashed rules. For that example, Tikz should be loaded (by `\usepackage{tikz}`).

```
\begin{pNiceMatrix}
\Block[borders={bottom,right,tikz=dashed}]{2-2}{
1 & 2 & 0 & 0 & 0 & 0 \\
4 & 5 & 0 & 0 & 0 & 0 \\
0 & 0 & \Block[borders={bottom,top,right,left,tikz=dashed}]{2-2}{
7 & 1 & 0 & 0 \\
0 & 0 & -1 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & \Block[borders={left,top,tikz=dashed}]{2-2}{
3 & 4 \\
0 & 0 & 0 & 0 & 1 & 4
\end{pNiceMatrix}
```

$$\left( \begin{array}{cccccc} 1 & 2 & 0 & 0 & 0 & 0 \\ 4 & 5 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 7 & 1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 4 \\ \hline 0 & 0 & 0 & 0 & 1 & 4 \end{array} \right)$$

## 18.6 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
$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 ;
0 64 -41 1 19 ;
0 0 0 0 0 { L_3 \gets 3 L_2 + L_3 }
\end{pNiceArray}$
```

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

```
\smallskip
```

```
$$\begin{pNiceArray}{rrrr|r}
```

```
12 -8 7 5 3 {} ;
```

```
0 64 -41 1 19 ;
```

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

```
\end{NiceMatrixBlock}
```

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

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}
```

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



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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

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

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

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

## 18.7 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>65</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}

```

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

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don't spread the cells of the array. We recall that, on the other side, the commands `\hline` and `\Hline`, the specifier “|” and the options `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` spread the cells.<sup>66</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.

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

```

\usepackage{tikz}
\usetikzlibrary{fit}

```

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

<sup>65</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

<sup>66</sup>For the command `\cline`, see the remark p. 10.

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

## 18.8 Utilisation of `\SubMatrix` in the `\CodeBefore`

In the following example, we illustrate the mathematical product of two matrices. The whole figure is an environment `{NiceArray}` and the three pairs of parenthesis have been added with `\SubMatrix` in the `\CodeBefore`.

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

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

[ \begin{NiceArray}{*{6}{c}@{\hspace{6mm}}*{5}{c}}[nullify-dots]
\CodeBefore [create-cell-nodes]
  \SubMatrix({2-7}{6-last})
  \SubMatrix({7-2}{last-6})
  \SubMatrix({7-7}{last-last})
  \begin{tikzpicture}
    \node [highlight = (9-2) (9-6)] { } ;
    \node [highlight = (2-9) (6-9)] { } ;
  \end{tikzpicture}
\Body
  & & & & & & & \color{blue}\scriptstyle C_j \\\
  & & & & & & b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\\
  & & & & & & \Vdots & & \Vdots & & \Vdots \\\
  & & & & & & & & b_{kj} \\\
  & & & & & & & & \Vdots \\\
  & & & & & & b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn} \\\[3mm]
  & a_{11} & \Cdots & & & a_{1n} \\\
  & \Vdots & & & & \Vdots & & & \Vdots \\\
\color{blue}\scriptstyle L_i
  & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} & \Cdots & & c_{ij} \\\
  & \Vdots & & & & \Vdots \\\
  & a_{n1} & \Cdots & & & a_{nn} \\\
\CodeAfter
\tikz \draw [gray,shorten > = 1mm, shorten < = 1mm] (9-4.north) to [bend left] (4-9.west) ;
\end{NiceArray}\]
```

## 19 Implementation

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

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

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

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

## Declaration of the package and packages loaded

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

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

`<@=@=nicematrix>`

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

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

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

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

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

We load some packages.

```
9 \RequirePackage { array }
10 \RequirePackage { amsmath }

11 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
12 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
13 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
14 \cs_generate_variant:Nn \@@_error:nn { n x }
15 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
16 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
17 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
18 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
```

With Overleaf, a document is compiled in non-stop mode. When there is an error, there is no way to the user to use the key `H` in order to have more information. That's why we decide to put that piece of information (for the messages with such information) in the main part of the message when the key `messages-for-Overleaf` is used (at load-time).

```
19 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
20   {
21     \bool_if:NTF \c_@@_messages_for_Overleaf_bool
22       { \msg_new:nnn { nicematrix } { #1 } { #2 } { #3 } }
23       { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
24   }
```

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

```
25 \cs_new_protected:Npn \@@_error_or_warning:n
26   { \bool_if:NTF \c_@@_messages_for_Overleaf_bool \@@_warning:n \@@_error:n }
```

We try to detect whether the compilation is done on Overleaf. We use `\c_sys_jobname_str` because, with Overleaf, the value of `\c_sys_jobname_str` is always “output”.

```

27 \bool_set:Nn \c_@@_messages_for_Overleaf_bool
28   {
29     \str_if_eq_p:Vn \c_sys_jobname_str { _region_ } % for Emacs
30     || \str_if_eq_p:Vn \c_sys_jobname_str { output } % for Overleaf
31   }

32 \cs_new_protected:Npn \@@_msg_redirect_name:nn
33   { \msg_redirect_name:nnn { nicematrix } }
34 \cs_new_protected:Npn \@@_gredirect_none:n #1
35   {
36     \group_begin:
37     \globaldefs = 1
38     \@@_msg_redirect_name:nn { #1 } { none }
39     \group_end:
40   }
41 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
42   {
43     \@@_error:n { #1 }
44     \@@_gredirect_none:n { #1 }
45   }
46 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
47   {
48     \@@_warning:n { #1 }
49     \@@_gredirect_none:n { #1 }
50   }

```

## Security test

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

However, if, one day, there is a changement in the implementation of `array`, maybe that this test will be broken (and `nicematrix` also).

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

```

51 \@@_msg_new:nn { Internal~error }
52   {
53     Potential~problem~when~using~nicematrix.\\
54     The~package~nicematrix~have~detected~a~modification~of~the~
55     standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
56     some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
57     this~message~again,~load~nicematrix~with:~\token_to_str:N
58     \usepackage[no-test-for-array]{nicematrix}.
59   }

60 \@@_msg_new:nn { mdwtab~loaded }
61   {
62     The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
63     This~error~is~fatal.
64   }

65 \cs_new_protected:Npn \@@_security_test:n #1
66   {

```

```

67 \peek_meaning:NTF \ignorespaces
68   { \@@_security_test_i:w }
69   { \@@_error:n { Internal~error } }
70   #1
71 }

72 \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
73 {
74   \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
75   #1
76 }

```

Here, the box `\l_tmpa_box` will be used as sandbox to take our security test.

```

77 \hook_gput_code:nnn { begindocument } { . }
78 {
79   \@ifpackageloaded { mdwtab }
80     { \@@_fatal:n { mdwtab-loaded } }
81     {
82       \@ifpackageloaded { fontspec }
83         { }
84         {
85           \bool_if:NF \c_@@_no_test_for_array_bool
86             {
87               \group_begin:
88                 \hbox_set:Nn \l_tmpa_box
89                   {
90                     \begin { tabular } { c > { \@@_security_test:n } c c }
91                     text & & text
92                     \end { tabular }
93                   }
94               \group_end:
95             }
96         }
97     }
98 }

```

## Technical definitions

```

99 \tl_new:N \l_@@_argspec_tl

100 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
101 \cs_generate_variant:Nn \keys_define:nn { n x }
102 \cs_generate_variant:Nn \str_lowercase:n { V }

103 \hook_gput_code:nnn { begindocument } { . }
104 {
105   \@ifpackageloaded { varwidth }
106     { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_true_bool } }
107     { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_false_bool } }
108   \@ifpackageloaded { booktabs }
109     { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_true_bool } }
110     { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_false_bool } }
111   \@ifpackageloaded { enumitem }
112     { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_true_bool } }
113     { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_false_bool } }
114   \@ifpackageloaded { tabularx }
115     { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_true_bool } }
116     { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_false_bool } }
117   \@ifpackageloaded { floatrow }
118     { \bool_const:Nn \c_@@_floatrow_loaded_bool { \c_true_bool } }

```



```

119     { \bool_const:Nn \c_@@_floatrow_loaded_bool { \c_false_bool } }
120     \@ifpackageloaded { tikz }
121     {

```

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

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

```

122     \bool_const:Nn \c_@@_tikz_loaded_bool \c_true_bool
123     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
124     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
125   }
126   {
127     \bool_const:Nn \c_@@_tikz_loaded_bool \c_false_bool
128     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
129     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
130   }
131 }

```

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 2022, the current version `revtex4-2` is 4.2e (compatible with `booktabs`).

```

132 \@ifclassloaded { revtex4-1 }
133   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
134   {
135     \@ifclassloaded { revtex4-2 }
136     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
137     {

```

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

```

138     \cs_if_exist:NT \rvtx@ifformat@geq
139     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
140     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
141   }
142 }

```

```

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

```

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

```

144 \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.

```

145 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
146   {
147     \iow_now:Nn \@mainaux
148     {
149       \ExplSyntaxOn
150       \cs_if_free:NT \pgfsyspdfmark
151       { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
152       \ExplSyntaxOff
153     }
154     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
155   }

```

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.

```

156 \ProvideDocumentCommand \iddots { }
157 {
158   \mathinner
159   {
160     \tex_mkern:D 1 mu
161     \box_move_up:nn { 1 pt } { \hbox:n { . } }
162     \tex_mkern:D 2 mu
163     \box_move_up:nn { 4 pt } { \hbox:n { . } }
164     \tex_mkern:D 2 mu
165     \box_move_up:nn { 7 pt }
166     { \vbox:n { \kern 7 pt \hbox:n { . } } }
167     \tex_mkern:D 1 mu
168   }
169 }

```

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.

```

170 \hook_gput_code:nnn { begindocument } { . }
171 {
172   \@ifpackageloaded { booktabs }
173   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
174   { }
175 }
176 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
177 {
178   \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`).

```

179   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
180   {
181     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
182     { \@_old_pgfulil@check@rerun { ##1 } { ##2 } }
183   }
184 }

```

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

```

185 \bool_new:N \l_@@_colortbl_loaded_bool
186 \hook_gput_code:nnn { begindocument } { . }
187 {
188   \@ifpackageloaded { colortbl }
189   { \bool_set_true:N \l_@@_colortbl_loaded_bool }
190   {

```

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.

```

191   \cs_set_protected:Npn \CT@arc@ { }
192   \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
193   \cs_set:Npn \CT@arc@ #1 #2
194   {
195     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
196     { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
197   }

```

Idem for `\CT@drs@`.

```

198   \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
199   \cs_set:Npn \CT@drs@ #1 #2

```

```

200     {
201       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
202         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
203     }
204   \cs_set:Npn \hline
205     {
206       \noalign { \ifnum 0 = ` } \fi
207       \cs_set_eq:NN \hskip \vskip
208       \cs_set_eq:NN \vrule \hrule
209       \cs_set_eq:NN \@width \@height
210       { \CT@arc@ \vline }
211       \futurelet \reserved@a
212       \@xhline
213     }
214   }
215 }

```

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.

```

216 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
217 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
218   {
219     \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
220     \int_compare:nNnT { #1 } > 1 { \multispan { \int_eval:n { #1 - 1 } } & }
221     \multispan { \int_eval:n { #2 - #1 + 1 } }
222     {
223       \CT@arc@
224       \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`<sup>67</sup>

```

225   \skip_horizontal:N \c_zero_dim
226 }

```

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`.

```

227   \everycr { }
228   \cr
229   \noalign { \skip_vertical:N -\arrayrulewidth }
230 }

```

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.

```

231 \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`.

```

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

```

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

```

233 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
234 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
235   {
236     \tl_if_empty:nTF { #3 }
237       { \@@_cline_iii:w #1|#2-#2 \q_stop }
238       { \@@_cline_ii:w #1|#2-#3 \q_stop }
239   }
240 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
241   { \@@_cline_iii:w #1|#2-#3 \q_stop }

```

<sup>67</sup>See question 99041 on TeX StackExchange.

```

242 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
243 {

```

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).

```

244 \int_compare:nNnT { #1 } < { #2 }
245 { \multispan { \int_eval:n { #2 - #1 } } & }
246 \multispan { \int_eval:n { #3 - #2 + 1 } }
247 {
248 \CT@arc@
249 \leaders \hrule \@height \arrayrulewidth \hfill
250 \skip_horizontal:N \c_zero_dim
251 }

```

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

```

252 \peek_meaning_remove_ignore_spaces:NTF \cline
253 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
254 { \everycr { } \cr }
255 }
256 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

The following command is a small shortcut.

```

257 \cs_new:Npn \@@_math_toggle_token:
258 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

```

```

259 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
260 {
261 \tl_if_blank:nF { #1 }
262 {
263 \tl_if_head_eq_meaning:nNTF { #1 } [
264 { \cs_set:Npn \CT@arc@ { \color #1 } }
265 { \cs_set:Npn \CT@arc@ { \color { #1 } } }
266 ]
267 }
268 \cs_generate_variant:Nn \@@_set_CT@arc@:n { V }

```

```

269 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
270 {
271 \tl_if_head_eq_meaning:nNTF { #1 } [
272 { \cs_set:Npn \CT@drsc@ { \color #1 } }
273 { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
274 ]
275 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { V }

```

The following command must *not* be protected since it will be used to write instructions in the (internal) `\CodeBefore`.

```

276 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
277 {
278 \tl_if_head_eq_meaning:nNTF { #2 } [
279 { #1 #2 }
280 { #1 { #2 } }
281 ]
282 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N V }

```

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

```

283 \cs_new_protected:Npn \@@_color:n #1
284 {
285 \tl_if_blank:nF { #1 }
286 { \@@_exp_color_arg:Nn \color { #1 } }
287 }
288 \cs_generate_variant:Nn \@@_color:n { V }

```

```

289 \cs_set_eq:NN \@@_old_pgfpoinanchor \pgfpoinanchor

```

## 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.

```
290 \bool_new:N \l_@@_siunitx_loaded_bool
291 \hook_gput_code:nnn { begindocument } { . }
292 {
293   \ifpackageloaded { siunitx }
294     { \bool_set_true:N \l_@@_siunitx_loaded_bool }
295     { }
296 }
```

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

```
297 \hook_gput_code:nnn { begindocument } { . }
298 {
299   \bool_if:nTF { ! \l_@@_siunitx_loaded_bool }
300     { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
301     {
302       \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
303         {
304           \renewcommand*{\NC@rewrite@S}[1] []
305         }

```

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

```
306       \tl_if_empty:nTF { ##1 }
307         {
308           \@temptokena \exp_after:wN
309             { \tex_the:D \@temptokena \@@_S: }
310         }
311         {
312           \@temptokena \exp_after:wN
313             { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
314         }
315       \NC@find
316     }
317   }
318 }
319 }
```

```
320 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
321 {
322   \tl_set_rescan:Nno
323     #1
324     {
325       \char_set_catcode_other:N >
326       \char_set_catcode_other:N <
327     }
328   #1
329 }
```

## Parameters

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

```
330 \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).

```
331 \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.

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

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

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

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

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

If, in a tabular, there is a tabular note in a caption that must be composed *above* the tabular, we will store in `\l_@@_note_in_caption_int` the number of notes in that caption. It will be stored in the `aux` file.

```
337 \int_new:N \l_@@_note_in_caption_int
```

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

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

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

```
339 \dim_new:N \l_@@_col_width_dim
340 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
341 \int_new:N \g_@@_row_total_int
342 \int_new:N \g_@@_col_total_int
```

The following parameter will be used by `\@@_create_row_node`: to avoid to create the same row-node twice (at the end of the array).

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

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

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

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

```
345 \str_new:N \l_@@_hpos_cell_str
346 \str_set:Nn \l_@@_hpos_cell_str { c }
```

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

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

Idem for the mono-row blocks.

```
348 \dim_new:N \g_@@_blocks_ht_dim
349 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

```
351 \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.

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

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

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

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

```
355 \bool_new:N \g_@@_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}`, `{NiceTabular*}` or `{NiceTabularX}`, we will raise the following flag.

```
356 \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.

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

The following dimension will be used for the total width of composite rules (*total* means that the spaces on both sides are included).

```
358 \dim_new:N \l_@@_rule_width_dim
```

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

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

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

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

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

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

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

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

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

The following parameter corresponds to the key `columns-type` of the environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. and also the key `matrix / columns-type` of `\NiceMatrixOptions`. However, it does *not* contain the value provided by the final user. Indeed, a transformation is done in order to have a preamble (for the package array) which is `nicematrix`-aware. That transformation is done with the command `\@@_set_preamble:Nn`.

```

364 \tl_new:N \l_@@_columns_type_tl
365 \hook_gput_code:nnn { begindocument } { . }
366   { \@@_set_preamble:Nn \l_@@_columns_type_tl { c } }

367 \cs_new_protected:Npn \@@_test_if_math_mode:
368   {
369     \if_mode_math: \else:
370       \@@_fatal:n { Outside-math-mode }
371     \fi:
372   }

```

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

```

373 \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.

```

374 \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”.

```

375 \colorlet { nicematrix-last-col } { . }
376 \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`).

```

377 \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*.

```

378 \tl_new:N \g_@@_com_or_env_str
379 \tl_gset: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).

```

380 \cs_new:Npn \@@_full_name_env:
381   {
382     \str_if_eq:VnTF \g_@@_com_or_env_str { command }
383       { command \space \c_backslash_str \g_@@_name_env_str }
384     { environment \space \{ \g_@@_name_env_str \} }
385   }

```

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

```

386 \tl_new:N \g_nicematrix_code_after_tl
387 \bool_new:N \l_@@_in_code_after_bool

```

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

```

388 \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_@@_pre_code_after_tl` because we must take care of the order in which instructions stored in that parameters are executed.

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

```
390 \tl_new:N \g_nicematrix_code_before_tl
```

```
391 \tl_new:N \g_@@_pre_code_before_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.

```
392 \int_new:N \l_@@_old_iRow_int
```

```
393 \int_new:N \l_@@_old_jCol_int
```

The TeX counters `\c@iRow` and `\c@jCol` will be created in the beginning of `{NiceArrayWithDelims}` (if they don't exist previously).

The following sequence will contain the names (without backslash) of the commands created by `custom-line` by the key `command` or `ccommand` (commands used by the final user in order to draw horizontal rules).

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

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

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

The sum of the weights of all the X-columns in the preamble. The weight of a X-column is given as an optional argument between square brackets. The default value, of course, is 1.

```
396 \int_new:N \g_@@_total_X_weight_int
```

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

```
397 \bool_new:N \l_@@_X_columns_aux_bool
```

```
398 \dim_new:N \l_@@_X_columns_dim
```

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if `\Hdotsfor` is used in that column.

```
399 \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).

```
400 \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).

```
401 \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`).

```
402 \tl_new:N \l_@@_code_before_tl
403 \bool_new:N \l_@@_code_before_bool
```

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

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

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

```
405 \dim_new:N \l_@@_x_initial_dim
406 \dim_new:N \l_@@_y_initial_dim
407 \dim_new:N \l_@@_x_final_dim
408 \dim_new:N \l_@@_y_final_dim
```

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

```
409 \dim_zero_new:N \l_@@_tmpc_dim
410 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
411 \bool_new:N \g_@@_empty_cell_bool
```

The following boolean will be used to deal with the commands `\tabularnote` in the caption (command `\caption` or key `caption`).

```
412 \bool_new:N \g_@@_second_composition_bool
```

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

```
413 \dim_new:N \g_@@_width_last_col_dim
414 \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.

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

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

```
416 \seq_new:N \g_@@_pos_of_blocks_seq
```

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

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

```
417 \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.

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

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

```
419 \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`).

```
420 \seq_new:N \g_@@_submatrix_names_seq
```

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

```
421 \bool_new:N \l_@@_width_used_bool
```

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

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

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

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

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

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

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

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

The following sequence will be used when the command `\SubMatrix` is used in the `\CodeBefore` (and not in the `\CodeAfter`). It will contain the position of all the sub-matrices specified in the `\CodeBefore`. 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.

```
428 \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).

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

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

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

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

```
432 \seq_new:N \l_@@_tikz_seq
```

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

```
434 \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` and also the key `color` of the command `\RowStyle`.

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

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

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

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

```
437 \str_new:N \l_@@_hpos_block_str
438 \str_set:Nn \l_@@_hpos_block_str { c }
439 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

For the vertical position, the possible values are `c`, `t` and `b`. Of course, it would be interesting to program a key `T` and a key `B`.

```
440 \str_new:N \l_@@_vpos_of_block_str
441 \str_set:Nn \l_@@_vpos_of_block_str { c }
```

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

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

The following flag corresponds to the keys `vlines` and `hlines` of the command `\Block` (the key `hvlines` is the conjunction of both).

```
443 \bool_new:N \l_@@_vlines_block_bool
444 \bool_new:N \l_@@_hlines_block_bool
```

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

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

446 \dim_new:N \l_@@_submatrix_extra_height_dim
447 \dim_new:N \l_@@_submatrix_left_xshift_dim
448 \dim_new:N \l_@@_submatrix_right_xshift_dim
449 \clist_new:N \l_@@_hlines_clist
450 \clist_new:N \l_@@_vlines_clist
451 \clist_new:N \l_@@_submatrix_hlines_clist
452 \clist_new:N \l_@@_submatrix_vlines_clist
```

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

```
453 \bool_new:N \l_@@_dotted_bool
```

The following flag will be set to true during the composition of a caption specified (by the key `caption`).

```
454 \bool_new:N \l_@@_in_caption_bool
```

## Variables for the exterior rows and columns

The keys for the exterior rows and columns are `first-row`, `first-col`, `last-row` and `last-col`. However, internally, these keys are not coded in a similar way.

- **First row**

The integer `\l_@@_first_row_int` is the number of the first row of the array. The default value is 1, but, if the option `first-row` is used, the value will be 0.

```
455 \int_new:N \l_@@_first_row_int
456 \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.

```
457 \int_new:N \l_@@_first_col_int
458 \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).

```
459 \int_new:N \l_@@_last_row_int
460 \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>68</sup>

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

Idem for `\l_@@_last_col_without_value_bool`

```
462 \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.

```
463 \int_new:N \l_@@_last_col_int
464 \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.

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

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

## Some utilities

```
466 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
467 {
468 \tl_set:Nn \l_tmpa_tl { #1 }
469 \tl_set:Nn \l_tmpb_tl { #2 }
470 }
```

---

<sup>68</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.

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers.

```

471 \cs_new_protected:Npn \@@_expand_clist:N #1
472 {
473   \clist_if_in:NnF #1 { all }
474   {
475     \clist_clear:N \l_tmpa_clist
476     \clist_map_inline:Nn #1
477     {
478       \tl_if_in:nnTF { ##1 } { - }
479       { \@@_cut_on_hyphen:w ##1 \q_stop }
480       {
481         \tl_set:Nn \l_tmpa_tl { ##1 }
482         \tl_set:Nn \l_tmpb_tl { ##1 }
483       }
484       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
485       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
486     }
487     \tl_set_eq:NN #1 \l_tmpa_clist
488   }
489 }

```

## The command `\tabularnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It's also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that's mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That's why:
  - The number of tabular notes present in the caption will be written on the aux file and available in `\l_@@_note_in_caption_int`.
  - During the composition of the main tabular, the tabular notes will be numbered from `\l_@@_note_in_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`.
  - During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\l_@@_note_in_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`.
  - After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

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

```

490 \newcounter { tabularnote }

```

```

491 \seq_new:N \g_@@_notes_seq
492 \seq_new:N \g_@@_notes_in_caption_seq

```

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

```

493 \tl_new:N \g_@@_tabularnote_tl

```

We prepare the tools for the formatting of the references of the footnotes (in the tabular itself). There may have several references of footnote at the same point and we have to take into account that point.

```

494 \seq_new:N \l_@@_notes_labels_seq
495 \newcounter{nicematrix_draft}
496 \cs_new_protected:Npn \@@_notes_format:n #1
497 {
498   \setcounter { nicematrix_draft } { #1 }
499   \@@_notes_style:n { nicematrix_draft }
500 }

```

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

```

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

```

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

```

502 \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`.

```

503 \cs_new:Npn \@@_notes_label_in_list:n #1 { \textsuperscript { #1 } }

```

We define `\thetabularnote` because it will be used by LaTeX if the user want to reference a tabular which has been marked by a `\label`. The TeX group is for the case where the user has put an instruction such as `\color{red}` in `\@@_notes_style:n`.

```

504 \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`).

```

505 \hook_gput_code:nnn { begindocument } { . }
506 {
507   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
508   {
509     \NewDocumentCommand \tabularnote { m }
510     {
511       \@@_error_or_warning:n { enumitem~not~loaded }
512       \@@_gredirect_none:n { enumitem~not~loaded }
513     }
514   }
515   {

```

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.

```

516     \newlist { tabularnotes } { enumerate } { 1 }
517     \setlist [ tabularnotes ]
518     {
519       topsep = 0pt ,
520       noitemsep ,
521       leftmargin = * ,
522       align = left ,
523       labelsep = 0pt ,

```

```

524     label =
525     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
526   }
527 \newlist { tabularnotes* } { enumerate* } { 1 }
528 \setlist [ tabularnotes* ]
529   {
530     afterlabel = \nobreak ,
531     itemjoin = \quad ,
532     label =
533     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
534   }

```

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

```

535 \NewDocumentCommand \tabularnote { m }
536 {
537   \bool_if:nT { \cs_if_exist_p:N \@@_capttype || \l_@@_in_env_bool }
538   {
539     \bool_if:nTF { ! \l_@@_NiceTabular_bool && \l_@@_in_env_bool }
540     { \@@_error:n { tabularnote~forbidden } }
541     {
542       \bool_if:NTF \l_@@_in_caption_bool
543       { \@@_tabularnote_ii:n { #1 } }
544       { \@@_tabularnote_i:n { #1 } }
545     }
546   }
547 }

```

For the version in normal conditions, that is to say not in the key `caption`.

```

548 \cs_new_protected:Npn \@@_tabularnote_i:n #1
549 {

```

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

```

550 \int_zero:N \l_tmpa_int
551 \bool_if:NT \l_@@_notes_detect_duplicates_bool
552 {
553   \seq_map_indexed_inline:Nn \g_@@_notes_seq
554   {
555     \tl_if_eq:nnT { #1 } { ##2 }
556     { \int_set:Nn \l_tmpa_int { ##1 } \seq_map_break: }
557   }
558   \int_compare:nNnF \l_tmpa_int = 0
559   { \int_add:Nn \l_tmpa_int \l_@@_note_in_caption_int }
560 }
561 \int_compare:nNnTF \l_tmpa_int = 0
562 {
563   \int_gincr:N \c@tabularnote
564   \seq_put_right:Nx \l_@@_notes_labels_seq
565   { \@@_notes_format:n { \int_use:c { c @ tabularnote } } }
566   \seq_gput_right:Nn \g_@@_notes_seq { #1 }
567 }
568 {
569   \seq_put_right:Nx \l_@@_notes_labels_seq
570   { \@@_notes_format:n { \int_use:N \l_tmpa_int } }
571 }
572 \peek_meaning:NF \tabularnote
573 {

```



If the following token is *not* a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell.

```
574         \hbox_set:Nn \l_tmpa_box
575         {
```

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

```
576         \@@_notes_label_in_tabular:n
577         {
578             \seq_use:Nnnn
579             \l_@@_notes_labels_seq { , } { , } { , }
580         }
581     }
```

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

```
582     \int_gsub:Nn \c@tabularnote { 1 }
583     \int_set_eq:NN \l_tmpa_int \c@tabularnote
584     \refstepcounter { tabularnote }
585     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
586     { \int_gincr:N \c@tabularnote }
587     \seq_clear:N \l_@@_notes_labels_seq
588     \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).

```
589         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
590     }
591 }
```

Now the version when the command is used in the key `caption`. The main difficulty is that the argument of the command `\caption` is composed several times. In order to know the number of commands `\tabularnote` in the caption, we will consider that there should not be the same tabular note twice in the caption (in the main tabular, it's possible). Once we have found a tabular note which has yet been encountered, we consider that you are in a new composition of the argument of `\caption`. At that time, we store in `\g_@@_nb_of_notes_int` the number of notes in the `\caption`.

```
592     \cs_new_protected:Npn \@@_tabularnote_ii:n #1
593     {
594         \int_gincr:N \c@tabularnote
595         \bool_if:NTF \g_@@_caption_finished_bool
596         {
597             \int_compare:nNnTF
598             \c@tabularnote > { \tl_count:N \g_@@_notes_in_caption_seq }
599             { \int_gset:Nn \c@tabularnote { 1 } }
600             \seq_if_in:NnF \g_@@_notes_in_caption_seq { #1 }
601             { \@@_fatal:n { Identical~notes~in~caption } }
602         }
603         {
604             \seq_if_in:NnTF \g_@@_notes_in_caption_seq { #1 }
605             {
606                 \bool_gset_true:N \g_@@_caption_finished_bool
607                 \int_gset:Nn \c@tabularnote { 1 }
608             }
609             { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { #1 } }
610         }
611         \seq_put_right:Nx \l_@@_notes_labels_seq
612         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
613         \peek_meaning:NF \tabularnote
614         {
615             \hbox_set:Nn \l_tmpa_box
616             {
```

```

617         \@@_notes_label_in_tabular:n
618         {
619             \seq_use:Nnnn
620             \l_@@_notes_labels_seq { , } { , } { , }
621         }
622     }
623     \seq_clear:N \l_@@_notes_labels_seq
624     \hbox_overlap_right:n { \box_use:N \l_tmpa_box }
625     \skip_horizontal:n { \box_wd:N \l_tmpa_box }
626 }
627 }
628 }
629 }

```

## 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.

```

630 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
631 {
632     \begin { pgfscope }
633     \pgfset
634     {
635         outer-sep = \c_zero_dim ,
636         inner-sep = \c_zero_dim ,
637         minimum-size = \c_zero_dim
638     }
639     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
640     \pgfnode
641     { rectangle }
642     { center }
643     {
644         \vbox_to_ht:nn
645         { \dim_abs:n { #5 - #3 } }
646         {
647             \vfill
648             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
649         }
650     }
651     { #1 }
652     { }
653     \end { pgfscope }
654 }

```

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.

```

655 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
656 {
657     \begin { pgfscope }
658     \pgfset
659     {
660         outer-sep = \c_zero_dim ,
661         inner-sep = \c_zero_dim ,
662         minimum-size = \c_zero_dim
663     }
664     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
665     \pgfpointdiff { #3 } { #2 }
666     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
667     \pgfnode
668     { rectangle }

```

```

669     { center }
670     {
671       \vbox_to_ht:nn
672         { \dim_abs:n \l_tmpb_dim }
673         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
674     }
675     { #1 }
676     { }
677   \end { pgfscope }
678 }

```

## The options

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

```

679 \tl_new:N \l_@@_caption_tl
680 \tl_new:N \l_@@_short_caption_tl
681 \tl_new:N \l_@@_label_tl

```

The following parameter corresponds to the key `caption-above` of `\NiceMatrixOptions`. When this parameter is `true`, the captions of the environments `{NiceTabular}`, specified with the key `caption` are put above the tabular (and below elsewhere).

```

682 \bool_new:N \l_@@_caption_above_bool

```

By default, the commands `\cellcolor` and `\rowcolor` are available for the user in the cells of the tabular (the user may use the commands provided by `\colortbl`). However, if the key `colortbl-like` is used, these commands are available.

```

683 \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 `\arrayrulewidth`. It's possible to disable this feature with the key `\l_@@_standard_line_bool`.

```

684 \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`).

```

685 \dim_new:N \l_@@_cell_space_top_limit_dim
686 \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.

```

687 \dim_new:N \l_@@_xdots_inter_dim
688 \hook_gput_code:nnn { begindocument } { . }
689 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }

```

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

The following dimension is the minimal distance between a node (in fact an anchor of that node) and a dotted line (we say “minimal” because, by definition, a dotted line is not a continuous line and, therefore, this distance may vary a little).

```

690 \dim_new:N \l_@@_xdots_shorten_start_dim
691 \dim_new:N \l_@@_xdots_shorten_end_dim
692 \hook_gput_code:nnn { begindocument } { . }
693 {
694   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
695   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
696 }

```

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

The following dimension is the radius of the dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.53 pt but it will be changed if the option `small` is used.

```
697 \dim_new:N \l_@@_xdots_radius_dim
698 \hook_gput_code:nnn { begindocument } { . }
699   { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }
```

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

The token list `\l_@@_xdots_line_style_tl` corresponds to the option `tikz` of the commands `\Cdots`, `\Ldots`, etc. and of the options `line-style` for the environments and `\NiceMatrixOptions`. The constant `\c_@@_standard_tl` will be used in some tests.

```
700 \tl_new:N \l_@@_xdots_line_style_tl
701 \tl_const:Nn \c_@@_standard_tl { standard }
702 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
703 \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).

```
704 \tl_new:N \l_@@_baseline_tl
705 \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`).

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

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

```
707 \bool_new:N \l_@@_parallelize_diags_bool
708 \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`.

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

```
710 \dim_new:N \l_@@_notes_above_space_dim
711 \hook_gput_code:nnn { begindocument } { . }
712   { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

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

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

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

The following flag corresponds to the key `respect-arraystretch` (that key has an effect on the blocks).

```
714 \bool_new:N \l_@@_respect_arraystretch_bool
```

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).

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

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

```
716 \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.

```
717 \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”.

```
718 \bool_new:N \l_@@_medium_nodes_bool
```

```
719 \bool_new:N \l_@@_large_nodes_bool
```

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

```
720 \bool_new:N \l_@@_except_borders_bool
```

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

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

```
722 \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`.

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

```
724 \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.

```
725 \tl_new:N \l_@@_end_of_row_tl
```

```
726 \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 “:”.

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

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

```
728 \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.

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

```

730 \keys_define:nn { NiceMatrix / xdots }
731 {
732   line-style .code:n =
733   {
734     \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.

```

735     { \cs_if_exist_p:N \tikzpicture }
736     { \str_if_eq_p:nn { #1 } { standard } }
737     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
738     { \@@_error:n { bad-option-for-line-style } }
739   } ,
740   line-style .value_required:n = true ,
741   color .tl_set:N = \l_@@_xdots_color_tl ,
742   color .value_required:n = true ,
743   shorten .code:n =
744     \hook_gput_code:nnn { begindocument } { . }
745     {
746       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
747       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
748     } ,
749   shorten-start .code:n =
750     \hook_gput_code:nnn { begindocument } { . }
751     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
752   shorten-end .code:n =
753     \hook_gput_code:nnn { begindocument } { . }
754     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,

```

We use a hook only by security in case `revtex4-1` is used (even though it is obsolete). Idem for the following keys.

```

755   shorten .value_required:n = true ,
756   shorten-start .value_required:n = true ,
757   shorten-end .value_required:n = true ,
758   radius .code:n =
759     \hook_gput_code:nnn { begindocument } { . }
760     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
761   radius .value_required:n = true ,
762   inter .code:n =
763     \hook_gput_code:nnn { begindocument } { . }
764     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
765   radius .value_required:n = true ,

```

The options `down` and `up` are not documented for the final user because he should use the syntax with `^` and `_`.

```

766   down .tl_set:N = \l_@@_xdots_down_tl ,
767   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).

```

768   draw-first .code:n = \prg_do_nothing: ,
769   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
770 }

```

```

771 \keys_define:nn { NiceMatrix / rules }
772 {
773   color .tl_set:N = \l_@@_rules_color_tl ,
774   color .value_required:n = true ,
775   width .dim_set:N = \arrayrulewidth ,
776   width .value_required:n = true ,
777   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
778 }

```

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

```

779 \keys_define:nn { NiceMatrix / Global }
780 {
781   custom-line .code:n = \l_@@_custom_line:n { #1 } ,
782   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
783   rules .value_required:n = true ,
784   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
785   standard-cline .default:n = true ,
786   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
787   cell-space-top-limit .value_required:n = true ,
788   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
789   cell-space-bottom-limit .value_required:n = true ,
790   cell-space-limits .meta:n =
791   {
792     cell-space-top-limit = #1 ,
793     cell-space-bottom-limit = #1 ,
794   } ,
795   cell-space-limits .value_required:n = true ,
796   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
797   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
798   light-syntax .default:n = true ,
799   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
800   end-of-row .value_required:n = true ,
801   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
802   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
803   last-row .int_set:N = \l_@@_last_row_int ,
804   last-row .default:n = -1 ,
805   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
806   code-for-first-col .value_required:n = true ,
807   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
808   code-for-last-col .value_required:n = true ,
809   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
810   code-for-first-row .value_required:n = true ,
811   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
812   code-for-last-row .value_required:n = true ,
813   hlines .clist_set:N = \l_@@_hlines_clist ,
814   vlines .clist_set:N = \l_@@_vlines_clist ,
815   hlines .default:n = all ,
816   vlines .default:n = all ,
817   vlines-in-sub-matrix .code:n =
818   {
819     \tl_if_single_token:nTF { #1 }
820     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
821     { \@@_error:n { One~letter~allowed } }
822   } ,
823   vlines-in-sub-matrix .value_required:n = true ,
824   hvlines .code:n =
825   {
826     \clist_set:Nn \l_@@_vlines_clist { all }
827     \clist_set:Nn \l_@@_hlines_clist { all }
828   } ,
829   hvlines-except-borders .code:n =
830   {
831     \clist_set:Nn \l_@@_vlines_clist { all }
832     \clist_set:Nn \l_@@_hlines_clist { all }
833     \bool_set_true:N \l_@@_except_borders_bool
834   } ,
835   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.

```

836 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
837 renew-dots .value_forbidden:n = true ,
838 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
839 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
840 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
841 create-extra-nodes .meta:n =
842   { create-medium-nodes , create-large-nodes } ,
843 left-margin .dim_set:N = \l_@@_left_margin_dim ,
844 left-margin .default:n = \arraycolsep ,
845 right-margin .dim_set:N = \l_@@_right_margin_dim ,
846 right-margin .default:n = \arraycolsep ,
847 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
848 margin .default:n = \arraycolsep ,
849 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
850 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
851 extra-margin .meta:n =
852   { extra-left-margin = #1 , extra-right-margin = #1 } ,
853 extra-margin .value_required:n = true ,
854 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
855 respect-arraystretch .default:n = true
856 }

```

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

```

857 \keys_define:nn { NiceMatrix / Env }
858   {
859     corners .clist_set:N = \l_@@_corners_clist ,
860     corners .default:n = { NW , SW , NE , SE } ,
861     code-before .code:n =
862       {
863         \tl_if_empty:nF { #1 }
864           {
865             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
866             \bool_set_true:N \l_@@_code_before_bool
867           }
868       } ,
869     code-before .value_required:n = true ,

```

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

```

870 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
871 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
872 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
873 baseline .tl_set:N = \l_@@_baseline_tl ,
874 baseline .value_required:n = true ,
875 columns-width .code:n =
876   \tl_if_eq:nnTF { #1 } { auto }
877     { \bool_set_true:N \l_@@_auto_columns_width_bool }
878     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
879 columns-width .value_required:n = true ,
880 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.

```

881 \legacy_if:nF { measuring@ }
882   {
883     \str_set:Nn \l_tmpa_str { #1 }
884     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
885       { \@@_error:nn { Duplicate-name } { #1 } }
886       { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
887     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
888   } ,

```



```

889 name .value_required:n = true ,
890 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
891 code-after .value_required:n = true ,
892 colortbl-like .code:n =
893   \bool_set_true:N \l_@@_colortbl_like_bool
894   \bool_set_true:N \l_@@_code_before_bool ,
895 colortbl-like .value_forbidden:n = true
896 }

897 \keys_define:nn { NiceMatrix / notes }
898 {
899   para .bool_set:N = \l_@@_notes_para_bool ,
900   para .default:n = true ,
901   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
902   code-before .value_required:n = true ,
903   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
904   code-after .value_required:n = true ,
905   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
906   bottomrule .default:n = true ,
907   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
908   style .value_required:n = true ,
909   label-in-tabular .code:n =
910     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
911   label-in-tabular .value_required:n = true ,
912   label-in-list .code:n =
913     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
914   label-in-list .value_required:n = true ,
915   enumitem-keys .code:n =
916     {
917       \hook_gput_code:nnn { begindocument } { . }
918       {
919         \bool_if:NT \c_@@_enumitem_loaded_bool
920           { \setlist* [ tabularnotes ] { #1 } }
921       }
922     } ,
923   enumitem-keys .value_required:n = true ,
924   enumitem-keys-para .code:n =
925     {
926       \hook_gput_code:nnn { begindocument } { . }
927       {
928         \bool_if:NT \c_@@_enumitem_loaded_bool
929           { \setlist* [ tabularnotes* ] { #1 } }
930       }
931     } ,
932   enumitem-keys-para .value_required:n = true ,
933   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
934   detect-duplicates .default:n = true ,
935   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
936 }

937 \keys_define:nn { NiceMatrix / delimiters }
938 {
939   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
940   max-width .default:n = true ,
941   color .tl_set:N = \l_@@_delimiters_color_tl ,
942   color .value_required:n = true ,
943 }

```

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

```

944 \keys_define:nn { NiceMatrix }
945 {
946   NiceMatrixOptions .inherit:n =
947     { NiceMatrix / Global } ,

```

```

948 NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
949 NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
950 NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
951 NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
952 SubMatrix / rules .inherit:n = NiceMatrix / rules ,
953 CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
954 CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
955 NiceMatrix .inherit:n =
956   {
957     NiceMatrix / Global ,
958     NiceMatrix / Env ,
959   } ,
960 NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
961 NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
962 NiceTabular .inherit:n =
963   {
964     NiceMatrix / Global ,
965     NiceMatrix / Env
966   } ,
967 NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
968 NiceTabular / rules .inherit:n = NiceMatrix / rules ,
969 NiceTabular / notes .inherit:n = NiceMatrix / notes ,
970 NiceArray .inherit:n =
971   {
972     NiceMatrix / Global ,
973     NiceMatrix / Env ,
974   } ,
975 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
976 NiceArray / rules .inherit:n = NiceMatrix / rules ,
977 pNiceArray .inherit:n =
978   {
979     NiceMatrix / Global ,
980     NiceMatrix / Env ,
981   } ,
982 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
983 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
984 }

```

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

```

985 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
986   {
987     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
988     delimiters / color .value_required:n = true ,
989     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
990     delimiters / max-width .default:n = true ,
991     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
992     delimiters .value_required:n = true ,
993     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
994     width .value_required:n = true ,
995     last-col .code:n =
996       \tl_if_empty:nF { #1 }
997         { \@@_error:n { last-col-non~empty-for~NiceMatrixOptions } }
998       \int_zero:N \l_@@_last_col_int ,
999     small .bool_set:N = \l_@@_small_bool ,
1000    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.

```

1001    renew-matrix .code:n = \@@_renew_matrix: ,
1002    renew-matrix .value_forbidden:n = true ,

```

The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

```
1003   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.

```
1004   columns-width .code:n =
1005     \tl_if_eq:nnTF { #1 } { auto }
1006     { \@@_error:n { Option~auto~for~columns~width } }
1007     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

```
1008   allow-duplicate-names .code:n =
1009     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1010   allow-duplicate-names .value_forbidden:n = true ,
1011   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1012   notes .value_required:n = true ,
1013   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1014   sub-matrix .value_required:n = true ,
1015   matrix / columns-type .code:n =
1016     \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1017   matrix / columns-type .value_required:n = true ,
1018   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1019   caption-above .default:n = true ,
1020   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1021 } ,
```

`\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.

```
1022 \NewDocumentCommand \NiceMatrixOptions { m }
1023 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }
```

We finalise the definition of the set of keys “`NiceMatrix / NiceMatrix`”. That set of keys will be used by `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

```
1024 \keys_define:nn { NiceMatrix / NiceMatrix }
1025 {
1026   last-col .code:n = \tl_if_empty:nTF {#1}
1027     {
1028       \bool_set_true:N \l_@@_last_col_without_value_bool
1029       \int_set:Nn \l_@@_last_col_int { -1 }
1030     }
1031     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1032   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1033   columns-type .value_required:n = true ,
1034   l .meta:n = { columns-type = l } ,
1035   r .meta:n = { columns-type = r } ,
1036   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1037   delimiters / color .value_required:n = true ,
1038   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1039   delimiters / max-width .default:n = true ,
1040   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1041   delimiters .value_required:n = true ,
1042   small .bool_set:N = \l_@@_small_bool ,
1043   small .value_forbidden:n = true ,
1044   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1045 }
```

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

```
1046 \keys_define:nn { NiceMatrix / NiceArray }
1047   {
```

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.

```
1048     small .bool_set:N = \l_@@_small_bool ,
1049     small .value_forbidden:n = true ,
1050     last-col .code:n = \tl_if_empty:nF { #1 }
1051                   { \@@_error:n { last-col~non~empty~for~NiceArray } }
1052                   \int_zero:N \l_@@_last_col_int ,
1053     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1054     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1055     unknown .code:n = \@@_error:n { Unknown-key-for~NiceArray }
1056   }
```

```
1057 \keys_define:nn { NiceMatrix / pNiceArray }
1058   {
1059     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1060     last-col .code:n = \tl_if_empty:nF {#1}
1061                   { \@@_error:n { last-col~non~empty~for~NiceArray } }
1062                   \int_zero:N \l_@@_last_col_int ,
1063     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1064     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1065     delimiters / color .value_required:n = true ,
1066     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1067     delimiters / max-width .default:n = true ,
1068     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1069     delimiters .value_required:n = true ,
1070     small .bool_set:N = \l_@@_small_bool ,
1071     small .value_forbidden:n = true ,
1072     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1073     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1074     unknown .code:n = \@@_error:n { Unknown-key-for~NiceMatrix }
1075   }
```

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

```
1076 \keys_define:nn { NiceMatrix / NiceTabular }
1077   {
```

The dimension width will be used if at least a column of type X is used. If there is no column of type X, an error will be raised.

```
1078     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1079                   \bool_set_true:N \l_@@_width_used_bool ,
1080     width .value_required:n = true ,
1081     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1082     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1083     tabularnote .value_required:n = true ,
1084     caption .tl_set:N = \l_@@_caption_tl ,
1085     caption .value_required:n = true ,
1086     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1087     short-caption .value_required:n = true ,
1088     label .tl_set:N = \l_@@_label_tl ,
1089     label .value_required:n = true ,
1090     last-col .code:n = \tl_if_empty:nF {#1}
1091                   { \@@_error:n { last-col~non~empty~for~NiceArray } }
1092                   \int_zero:N \l_@@_last_col_int ,
1093     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1094     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1095     unknown .code:n = \@@_error:n { Unknown-key-for~NiceTabular }
1096   }
```

## Important code used by `{NiceArrayWithDelims}`

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

```
1097 \cs_new_protected:Npn \@@_cell_begin:w
1098   {
```

`\g_@@_cell_after_hook_tl` will be set during the composition of the box `\l_@@_cell_box` and will be used *after* the composition in order to modify that box.

```
1099   \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

```
1100   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1101   \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 `arydshn`, create special rows in the `\halign` that we don't want to take into account.

```
1102   \int_compare:nNnT \c@jCol = 1
1103     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
```

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the `\@@_cell_end:` (and the potential `\c_math_toggle_token` also).

```
1104   \hbox_set:Nw \l_@@_cell_box
1105   \bool_if:NF \l_@@_NiceTabular_bool
1106     {
1107       \c_math_toggle_token
1108       \bool_if:NT \l_@@_small_bool \scriptstyle
1109     }
1110   \g_@@_row_style_tl
```

We will call *corners* of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn't always exist simultaneously).

The codes `\l_@@_code_for_first_row_tl` and `al` don't apply in the corners of the matrix.

```
1111   \int_compare:nNnTF \c@iRow = 0
1112     {
1113       \int_compare:nNnT \c@jCol > 0
1114         {
1115           \l_@@_code_for_first_row_tl
1116           \xglobal \colorlet { nicematrix-first-row } { . }
1117         }
1118     }
1119     {
1120       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1121         {
1122           \l_@@_code_for_last_row_tl
1123           \xglobal \colorlet { nicematrix-last-row } { . }
1124         }
1125     }
1126 }
```

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.

```
1127 \cs_new_protected:Npn \@@_begin_of_row:
1128   {
1129     \int_gincr:N \c@iRow
1130     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1131     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1132     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
```

```

1133 \pgfpicture
1134 \pgfrememberpicturepositiononpagetrue
1135 \pgfcoordinate
1136 { \@@_env: - row - \int_use:N \c@iRow - base }
1137 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1138 \str_if_empty:NF \l_@@_name_str
1139 {
1140   \pgfnodealias
1141   { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1142   { \@@_env: - row - \int_use:N \c@iRow - base }
1143 }
1144 \endpgfpicture
1145 }

```

Remark: If the key `recreate-cell-nodes` of the `\CodeBefore` is used, then we will add some lines to that command.

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command.

```

1146 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1147 {
1148   \int_compare:nNnTF \c@iRow = 0
1149   {
1150     \dim_gset:Nn \g_@@_dp_row_zero_dim
1151     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1152     \dim_gset:Nn \g_@@_ht_row_zero_dim
1153     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1154   }
1155   {
1156     \int_compare:nNnT \c@iRow = 1
1157     {
1158       \dim_gset:Nn \g_@@_ht_row_one_dim
1159       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1160     }
1161   }
1162 }
1163 \cs_new_protected:Npn \@@_rotate_cell_box:
1164 {
1165   \box_rotate:Nn \l_@@_cell_box { 90 }
1166   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1167   {
1168     \vbox_set_top:Nn \l_@@_cell_box
1169     {
1170       \vbox_to_zero:n { }
1171       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1172       \box_use:N \l_@@_cell_box
1173     }
1174   }
1175   \bool_gset_false:N \g_@@_rotate_bool
1176 }
1177 \cs_new_protected:Npn \@@_adjust_size_box:
1178 {
1179   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1180   {
1181     \box_set_wd:Nn \l_@@_cell_box
1182     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1183     \dim_gzero:N \g_@@_blocks_wd_dim
1184   }
1185   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1186   {
1187     \box_set_dp:Nn \l_@@_cell_box

```

```

1188     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1189     \dim_gzero:N \g_@@_blocks_dp_dim
1190   }
1191   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1192   {
1193     \box_set_ht:Nn \l_@@_cell_box
1194     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1195     \dim_gzero:N \g_@@_blocks_ht_dim
1196   }
1197 }
1198 \cs_new_protected:Npn \@@_cell_end:
1199 {
1200   \@@_math_toggle_token:
1201   \hbox_set_end:

```

The token list `\g_@@_cell_after_hook_tl` is (potentially) set during the composition of the box `\l_@@_cell_box` and is used now *after* the composition in order to modify that box.

```

1202   \g_@@_cell_after_hook_tl
1203   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1204   \@@_adjust_size_box:
1205   \box_set_ht:Nn \l_@@_cell_box
1206   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1207   \box_set_dp:Nn \l_@@_cell_box
1208   { \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”).

```

1209   \dim_gset:Nn \g_@@_max_cell_width_dim
1210   { \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”.

```

1211   \@@_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:

- for the columns of type p, m, b, V (of `varwidth`) or X, we test whether the cell is syntactically empty with `\@@_test_if_empty:` and `\@@_test_if_empty_for_S:`
- if the width of the box `\l_@@_cell_box` (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a `\rlap`, `\llap`, `\clap` or a `\mathclap` of `mathtools`).
- the cells with a command `\Ldots` or `\Cdots`, `\Vdots`, etc., should also be considered as empty; if `nullify-dots` is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of `\CodeAfter`); however, if `nullify-dots` is not in force, a phantom of `\ldots`, `\cdots`, `\vdots` is inserted and its width is not equal to zero; that’s why these commands raise a boolean `\g_@@_empty_cell_bool` and we begin by testing this boolean.

```

1212   \bool_if:NTF \g_@@_empty_cell_bool
1213   { \box_use_drop:N \l_@@_cell_box }
1214   {
1215     \bool_lazy_or:nnTF
1216     \g_@@_not_empty_cell_bool
1217     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1218     \@@_node_for_cell:
1219     { \box_use_drop:N \l_@@_cell_box }

```

```

1220     }
1221     \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1222     \bool_gset_false:N \g_@@_empty_cell_bool
1223     \bool_gset_false:N \g_@@_not_empty_cell_bool
1224 }

```

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

```

1225 \cs_new_protected:Npn \@@_node_for_cell:
1226 {
1227     \pgfpicture
1228     \pgfsetbaseline \c_zero_dim
1229     \pgfrememberpicturepositiononpagetrue
1230     \pgfset
1231     {
1232         inner-sep = \c_zero_dim ,
1233         minimum-width = \c_zero_dim
1234     }
1235     \pgfnode
1236     { rectangle }
1237     { base }
1238     {

```

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

```

1239         \set@color
1240         \box_use_drop:N \l_@@_cell_box
1241     }
1242     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1243     { }
1244     \str_if_empty:NF \l_@@_name_str
1245     {
1246         \pgfnodealias
1247         { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1248         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1249     }
1250     \endpgfpicture
1251 }

```

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.

```

1252 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1253 {
1254     \cs_new_protected:Npn \@@_patch_node_for_cell:
1255     {
1256         \hbox_set:Nn \l_@@_cell_box
1257         {
1258             \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1259             \hbox_overlap_left:n
1260             {
1261                 \pgfsys@markposition
1262                 { \@@_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.

```

1263         #1
1264     }
1265     \box_use:N \l_@@_cell_box
1266     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1267     \hbox_overlap_left:n
1268     {
1269         \pgfsys@markposition
1270         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }

```



```

1271         #1
1272     }
1273 }
1274 }
1275 }

```

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

```

1276 \bool_lazy_or:nTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1277 {
1278     \@@_patch_node_for_cell:n
1279     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1280 }
1281 { \@@_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}

```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 \dots\dots\dots 6 \\ 7 \dots\dots\dots \end{pmatrix}$$

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.

```

1282 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1283 {
1284     \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1285     { g_@@_#2 _ lines _ tl }
1286     {
1287         \use:c { @@ _ draw _ #2 : nnn }
1288         { \int_use:N \c@iRow }
1289         { \int_use:N \c@jCol }
1290         { \exp_not:n { #3 } }
1291     }
1292 }

1293 \cs_new_protected:Npn \@@_array:n
1294 {
1295     \bool_if:NTF \l_@@_NiceTabular_bool
1296     { \dim_set_eq:NN \col@sep \tabcolsep }
1297     { \dim_set_eq:NN \col@sep \arraycolsep }
1298     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1299     { \cs_set_nopar:Npn \@halignto { } }
1300     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1301     \@tabarray

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

1302     [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1303 }
1304 \cs_generate_variant:Nn \@@_array:n { V }

```

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.

```
1305 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1306 \cs_new_protected:Npn \@@_create_row_node:
1307 {
1308   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1309   {
1310     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1311     \@@_create_row_node_i:
1312   }
1313 }
1314 \cs_new_protected:Npn \@@_create_row_node_i:
1315 {
```

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

```
1316   \hbox
1317   {
1318     \bool_if:NT \l_@@_code_before_bool
1319     {
1320       \vtop
1321       {
1322         \skip_vertical:N 0.5\arrayrulewidth
1323         \pgfsys@markposition
1324         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1325         \skip_vertical:N -0.5\arrayrulewidth
1326       }
1327     }
1328     \pgfpicture
1329     \pgfrememberpicturepositiononpagetrue
1330     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1331     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1332     \str_if_empty:NF \l_@@_name_str
1333     {
1334       \pgfnodealias
1335       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1336       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1337     }
1338     \endpgfpicture
1339   }
1340 }
```

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

```
1341 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1342 \cs_new_protected:Npn \@@_everycr_i:
1343 {
1344   \int_gzero:N \c@jCol
1345   \bool_gset_false:N \g_@@_after_col_zero_bool
1346   \bool_if:NF \g_@@_row_of_col_done_bool
1347   {
1348     \@@_create_row_node:
```

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

```
1349   \tl_if_empty:NF \l_@@_hlines_clist
1350   {
1351     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1352     {
1353       \exp_args:NNx
1354       \clist_if_in:NnT
1355       \l_@@_hlines_clist
```

```

1356         { \int_eval:n { \c@iRow + 1 } }
1357     }
1358     {

```

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.

```

1359         \int_compare:nNnT \c@iRow > { -1 }
1360     {
1361         \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@`.

```

1362         { \hrule height \arrayrulewidth width \c_zero_dim }
1363     }
1364 }
1365 }
1366 }
1367 }

```

The command `\@@_newcolumnntype` is the command `\newcolumnntype` of `array` without the warnings for redefinitions of columns types (we will use it to redefine the columns types `w` and `W`).

```

1368 \cs_set_protected:Npn \@@_newcolumnntype #1
1369 {
1370     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1371     \peek_meaning:NTF [
1372         { \newcol@ #1 }
1373         { \newcol@ #1 [ 0 ] }
1374     ]

```

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

```

1375 \cs_set_protected:Npn \@@_renew_dots:
1376 {
1377     \cs_set_eq:NN \ldots \@@_Ldots
1378     \cs_set_eq:NN \cdots \@@_Cdots
1379     \cs_set_eq:NN \vdots \@@_Vdots
1380     \cs_set_eq:NN \ddots \@@_Ddots
1381     \cs_set_eq:NN \iddots \@@_Iddots
1382     \cs_set_eq:NN \dots \@@_Ldots
1383     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1384 }

```

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

```

1385 \cs_new_protected:Npn \@@_colortbl_like:
1386 {
1387     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1388     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1389     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1390 }

```

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

```

1391 \cs_new_protected:Npn \@@_pre_array_ii:
1392 {

```

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

```

1393     \int_gzero:N \g_@@_total_X_weight_int
1394     \@@_expand_clist:N \l_@@_hlines_clist
1395     \@@_expand_clist:N \l_@@_vlines_clist

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` before the vertical skip (and thus, at a wrong place). That why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition<sup>69</sup>.

```

1396   \bool_if:NT \c_@@_booktabs_loaded_bool
1397     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1398   \box_clear_new:N \l_@@_cell_box
1399   \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}`).

```

1400   \bool_if:NT \l_@@_small_bool
1401     {
1402       \cs_set_nopar:Npn \arraystretch { 0.47 }
1403       \dim_set:Nn \arraycolsep { 1.45 pt }
1404     }

1405   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1406     {
1407       \tl_put_right:Nn \@@_begin_of_row:
1408         {
1409           \pgfsys@markposition
1410             { \@@_env: - row - \int_use:N \c@iRow - base }
1411         }
1412     }

```

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`.

```

1413   \cs_set_nopar:Npn \ialign
1414     {
1415       \bool_if:NTF \l_@@_colortbl_loaded_bool
1416         {
1417           \CT@everycr
1418             {
1419               \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1420               \@@_everycr:
1421             }
1422         }
1423       { \everycr { \@@_everycr: } }
1424       \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>70</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`.

```

1425   \dim_gzero_new:N \g_@@_dp_row_zero_dim
1426   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1427   \dim_gzero_new:N \g_@@_ht_row_zero_dim
1428   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }

```

<sup>69</sup>cf. `\nicematrix@redefine@check@rerun`

<sup>70</sup>The option `small` of `nicematrix` changes (among others) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1429     \dim_gzero_new:N \g_@@_ht_row_one_dim
1430     \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1431     \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1432     \dim_gzero_new:N \g_@@_ht_last_row_dim
1433     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1434     \dim_gzero_new:N \g_@@_dp_last_row_dim
1435     \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`.

```

1436     \cs_set_eq:NN \ialign \@@_old_ialign:
1437     \halign
1438 }

```

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).

```

1439     \cs_set_eq:NN \@@_old_ldots \ldots
1440     \cs_set_eq:NN \@@_old_cdots \cdots
1441     \cs_set_eq:NN \@@_old_vdots \vdots
1442     \cs_set_eq:NN \@@_old_ddots \ddots
1443     \cs_set_eq:NN \@@_old_iddots \iddots
1444     \bool_if:NTF \l_@@_standard_cline_bool
1445     { \cs_set_eq:NN \cline \@@_standard_cline }
1446     { \cs_set_eq:NN \cline \@@_cline }
1447     \cs_set_eq:NN \Ldots \@@_Ldots
1448     \cs_set_eq:NN \Cdots \@@_Cdots
1449     \cs_set_eq:NN \Vdots \@@_Vdots
1450     \cs_set_eq:NN \Ddots \@@_Ddots
1451     \cs_set_eq:NN \Iddots \@@_Iddots
1452     \cs_set_eq:NN \Hline \@@_Hline:
1453     \cs_set_eq:NN \Hspace \@@_Hspace:
1454     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1455     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1456     \cs_set_eq:NN \Block \@@_Block:
1457     \cs_set_eq:NN \rotate \@@_rotate:
1458     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1459     \cs_set_eq:NN \dotfill \@@_old_dotfill:
1460     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1461     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1462     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1463     \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1464     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1465     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1466     \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1467     \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition.

```

1468     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1469     \hook_gput_code:nnn { env / tabular / begin } { . }
1470     { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }

```

If there is one or several commands `\tabularnote` in the caption specified by the key `caption` and if that caption has to be composed above the `tabular`, we have now that information because it has been written in the aux file at a previous run. We use that information to start counting the `tabular` notes in the main array at the right value (that remember that the caption will be composed *after* the array!).

```

1471     \tl_if_exist:NT \l_@@_note_in_caption_tl
1472     {
1473         \tl_if_empty:NF \l_@@_note_in_caption_tl
1474         {
1475             \int_set_eq:NN \l_@@_note_in_caption_int

```

```

1476         { \l_@@_note_in_caption_tl }
1477         \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1478     }
1479 }

```

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

```

1480     \seq_gclear:N \g_@@_multicolumn_cells_seq
1481     \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

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

```

1482     \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`).

```

1483     \int_gzero_new:N \g_@@_row_total_int

```

The counter `\c@jCol` will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter `\g_@@_col_total_int`. These counters are updated in the command `\@@_cell_begin:w` executed at the beginning of each cell.

```

1484     \int_gzero_new:N \g_@@_col_total_int
1485     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1486     \@@_renew_NC@rewrite@S:
1487     \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.

```

1488     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1489     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1490     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1491     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1492     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1493     \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1494     \tl_gclear:N \g_nicematrix_code_before_tl
1495     \tl_gclear:N \g_@@_pre_code_before_tl
1496 }

```

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

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

```

1497 \cs_new_protected:Npn \@@_pre_array:
1498 {
1499     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1500     \int_gzero_new:N \c@iRow
1501     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1502     \int_gzero_new:N \c@jCol

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_int` are *not* the numbers of the last row and last column of the array. There are only the values of the keys `last-row` and `last-column` (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of `nicematrix`. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the `aux` file (of course, it’s possible only after the first compilation).

```

1503     \int_compare:nNnT \l_@@_last_row_int = { -1 }
1504     {
1505         \bool_set_true:N \l_@@_last_row_without_value_bool

```

```

1506     \bool_if:NT \g_@@_aux_found_bool
1507     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1508   }
1509   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1510   {
1511     \bool_if:NT \g_@@_aux_found_bool
1512     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1513   }

```

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

```

1514   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1515   {
1516     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1517     {
1518       \dim_gset:Nn \g_@@_ht_last_row_dim
1519       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1520       \dim_gset:Nn \g_@@_dp_last_row_dim
1521       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1522     }
1523   }

1524   \seq_gclear:N \g_@@_cols_vlism_seq
1525   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1526   \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

The value of `\g_@@_pos_of_blocks_seq` has been written on the `aux` file and loaded before the (potential) execution of the `\CodeBefore`. Now, we clear that variable because it will be reconstructed during the creation of the array.

```

1527   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1528   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1529   \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1530   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1531   \@@_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.

```

1532   \box_clear_new:N \l_@@_the_array_box

```

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

```

1533 \dim_zero_new:N \l_@@_left_delim_dim
1534 \dim_zero_new:N \l_@@_right_delim_dim
1535 \bool_if:NTF \g_@@_NiceArray_bool
1536 {
1537   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1538   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1539 }
1540 {

```

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

```

1541 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1542 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1543 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1544 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1545 }

```

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).

```

1546 \hbox_set:Nw \l_@@_the_array_box
1547 \skip_horizontal:N \l_@@_left_margin_dim
1548 \skip_horizontal:N \l_@@_extra_left_margin_dim
1549 \c_math_toggle_token
1550 \bool_if:NTF \l_@@_light_syntax_bool
1551 { \use:c { @@-light-syntax } }
1552 { \use:c { @@-normal-syntax } }
1553 }

```

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

```

1554 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1555 {
1556   \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1557   \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...

```

1558 \@@_pre_array:
1559 }

```

## The `\CodeBefore`

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

```

1560 \cs_new_protected:Npn \@@_pre_code_before:
1561 {

```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1562 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1563 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1564 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1565 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }

```



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

```

1566 \pgfsys@markposition { \@@_env: - position }
1567 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1568 \pgfpicture
1569 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1570 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1571 {
1572   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1573   \pgfcoordinate { \@@_env: - row - ##1 }
1574   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1575 }

```

Now, the recreation of the col nodes.

```

1576 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1577 {
1578   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1579   \pgfcoordinate { \@@_env: - col - ##1 }
1580   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1581 }

```

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

```

1582 \@@_create_diag_nodes:

```

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

```

1583 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1584 \endpgfpicture

```

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

```

1585 \@@_create_blocks_nodes:
1586 \bool_if:NT \c_@@_tikz_loaded_bool
1587 {
1588   \tikzset
1589   {
1590     every~picture / .style =
1591     { overlay , name~prefix = \@@_env: - }
1592   }
1593 }
1594 \cs_set_eq:NN \cellcolor \@@_cellcolor
1595 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1596 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1597 \cs_set_eq:NN \rowcolor \@@_rowcolor
1598 \cs_set_eq:NN \rowcolors \@@_rowcolors
1599 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1600 \cs_set_eq:NN \arraycolor \@@_arraycolor
1601 \cs_set_eq:NN \columncolor \@@_columncolor
1602 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1603 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1604 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1605 }

```

```

1606 \cs_new_protected:Npn \@@_exec_code_before:
1607 {
1608   \seq_gclear_new:N \g_@@_colors_seq
1609   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1610   \group_begin:

```

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

```

1611 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token

```

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

```

1612     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1613     {
1614         \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1615         \@@_rescan_for_spanish:N \l_@@_code_before_tl
1616     }

```

Here is the `\CodeBefore`. The construction is a bit complicated because `\g_@@_pre_code_before_tl` may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do *not* execute the rest of `\g_@@_pre_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That's why we use a `\q_stop`: it will be used to discard the rest of `\g_@@_pre_code_before_tl`.

```

1617     \exp_last_unbraced:NV \@@_CodeBefore_keys:
1618     \g_@@_pre_code_before_tl

```

Now, all the cells which are specified to be colored by instructions in the `\CodeBefore` will actually be colored. It's a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

```

1619     \@@_actually_color:
1620     \l_@@_code_before_tl
1621     \q_stop
1622     \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1623     \group_end:
1624     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1625     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1626 }

```

```

1627 \keys_define:nn { NiceMatrix / CodeBefore }
1628 {
1629     create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1630     create-cell-nodes .default:n = true ,
1631     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1632     sub-matrix .value_required:n = true ,
1633     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1634     delimiters / color .value_required:n = true ,
1635     unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1636 }
1637 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1638 {
1639     \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1640     \@@_CodeBefore:w
1641 }

```

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.

```

1642 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1643 {
1644     \bool_if:NT \g_@@_aux_found_bool
1645     {
1646         \@@_pre_code_before:
1647         #1
1648     }
1649 }

```

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.

```

1650 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1651 {
1652   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1653   {
1654     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1655     \pgfcoordinate { \@@_env: - row - ##1 - base }
1656     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1657     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1658     {
1659       \cs_if_exist:cT
1660       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1661       {
1662         \pgfsys@getposition
1663         { \@@_env: - ##1 - ####1 - NW }
1664         \@@_node_position:
1665         \pgfsys@getposition
1666         { \@@_env: - ##1 - ####1 - SE }
1667         \@@_node_position_i:
1668         \@@_pgf_rect_node:nnn
1669         { \@@_env: - ##1 - ####1 }
1670         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1671         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1672       }
1673     }
1674   }
1675   \int_step_inline:nn \c@iRow
1676   {
1677     \pgfnodealias
1678     { \@@_env: - ##1 - last }
1679     { \@@_env: - ##1 - \int_use:N \c@jCol }
1680   }
1681   \int_step_inline:nn \c@jCol
1682   {
1683     \pgfnodealias
1684     { \@@_env: - last - ##1 }
1685     { \@@_env: - \int_use:N \c@iRow - ##1 }
1686   }
1687   \@@_create_extra_nodes:
1688 }

```

```

1689 \cs_new_protected:Npn \@@_create_blocks_nodes:
1690 {
1691   \pgfpicture
1692   \pgf@relevantforpicturesizefalse
1693   \pgfrememberpicturepositiononpagetrue
1694   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1695   { \@@_create_one_block_node:nnnnn ##1 }
1696   \endpgfpicture
1697 }

```

The following command is called `\@@_create_one_block_node:nnnnn` but, in fact, it creates a node only if the last argument (#5) which is the name of the block, is not empty.<sup>71</sup>

```

1698 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1699 {
1700   \tl_if_empty:nF { #5 }
1701   {
1702     \@@_qpoint:n { col - #2 }
1703     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1704     \@@_qpoint:n { #1 }

```

<sup>71</sup>Moreover, there is also in the list `\g_@@_pos_of_blocks_seq` the positions of the dotted lines (created by `\Cdots`, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).

```

1705     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1706     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1707     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1708     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1709     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1710     \@@_pgf_rect_node:nnnnn
1711     { \@@_env: - #5 }
1712     { \dim_use:N \l_tmpa_dim }
1713     { \dim_use:N \l_tmpb_dim }
1714     { \dim_use:N \l_@@_tmpc_dim }
1715     { \dim_use:N \l_@@_tmpd_dim }
1716   }
1717 }

1718 \cs_new_protected:Npn \@@_patch_for_revtext:
1719 {
1720   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1721   \cs_set_eq:NN \insert@column \insert@column@array
1722   \cs_set_eq:NN \@classx \@classx@array
1723   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1724   \cs_set_eq:NN \@arraycr \@arraycr@array
1725   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1726   \cs_set_eq:NN \array \array@array
1727   \cs_set_eq:NN \@array \@array@array
1728   \cs_set_eq:NN \@tabular \@tabular@array
1729   \cs_set_eq:NN \@mkpream \@mkpream@array
1730   \cs_set_eq:NN \endarray \endarray@array
1731   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1732   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1733 }

```

## The environment {NiceArrayWithDelims}

```

1734 \NewDocumentEnvironment { NiceArrayWithDelims }
1735 { m m O { } m ! O { } t \CodeBefore }
1736 {
1737   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1738   \@@_provide_pgfsyspdfmark:
1739   \bool_if:NT \c_@@_footnote_bool \savenotes

```

The aim of the following `\bgroup` (the corresponding `\egroup` is, of course, at the end of the environment) is to be able to put an exponent to a matrix in a mathematical formula.

```

1740   \bgroup

1741   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1742   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1743   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1744   \int_gzero:N \g_@@_block_box_int
1745   \dim_zero:N \g_@@_width_last_col_dim
1746   \dim_zero:N \g_@@_width_first_col_dim
1747   \bool_gset_false:N \g_@@_row_of_col_done_bool
1748   \str_if_empty:NT \g_@@_name_env_str
1749     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1750   \bool_if:NTF \l_@@_NiceTabular_bool
1751     \mode_leave_vertical:
1752     \@@_test_if_math_mode:
1753   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1754   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>72</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.

```
1755 \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.

```
1756 \cs_if_exist:NT \tikz@library@external@loaded
1757 {
1758   \tikzexternaldisable
1759   \cs_if_exist:NT \ifstandalone
1760   { \tikzset { external / optimize = false } }
1761 }
```

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

```
1762 \int_gincr:N \g_@@_env_int
1763 \bool_if:NF \l_@@_block_auto_columns_width_bool
1764 { \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 (except the blocks with the key `hvlines`).

```
1765 \seq_gclear:N \g_@@_blocks_seq
1766 \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`.

```
1767 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1768 \seq_gclear:N \g_@@_pos_of_xdots_seq
1769 \tl_gclear_new:N \g_@@_code_before_tl
1770 \tl_gclear:N \g_@@_row_style_tl
```

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

```
1771 \bool_gset_false:N \g_@@_aux_found_bool
1772 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1773 {
1774   \bool_gset_true:N \g_@@_aux_found_bool
1775   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1776 }
```

Now, we prepare the token list for the instructions that we will have to write on the aux file at the end of the environment.

```
1777 \tl_gclear:N \g_@@_aux_tl
1778 \tl_if_empty:NF \g_@@_code_before_tl
1779 {
1780   \bool_set_true:N \l_@@_code_before_bool
1781   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1782 }
1783 \tl_if_empty:NF \g_@@_pre_code_before_tl
1784 { \bool_set_true:N \l_@@_code_before_bool }
```

The set of keys is not exactly the same for `{NiceArray}` and for the variants of `{NiceArray}` (`{pNiceArray}`, `{bNiceArray}`, etc.) because, for `{NiceArray}`, we have the options `t`, `c`, `b` and `baseline`.

```
1785 \bool_if:NTF \g_@@_NiceArray_bool
1786 { \keys_set:nn { NiceMatrix / NiceArray } }
1787 { \keys_set:nn { NiceMatrix / pNiceArray } }
1788 { #3 , #5 }
```

---

<sup>72</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```
1789 \@@_set_CT@arc@:V \l_@@_rules_color_tl
```

The argument #6 is the last argument of {NiceArrayWithDelims}. With that argument of type “t \CodeBefore”, we test whether there is the keyword \CodeBefore at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword \CodeBefore and the (other) keyword \Body. It’s the job that will do the command \@@\_CodeBefore\_Body:w. After that job, the command \@@\_CodeBefore\_Body:w will go on with \@@\_pre\_array:.

```
1790 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1791 }
```

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

```
1792 {
1793   \bool_if:NTF \l_@@_light_syntax_bool
1794     { \use:c { end @@-light-syntax } }
1795     { \use:c { end @@-normal-syntax } }
1796   \c_math_toggle_token
1797   \skip_horizontal:N \l_@@_right_margin_dim
1798   \skip_horizontal:N \l_@@_extra_right_margin_dim
1799   \hbox_set_end:
```

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

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

```
1800 \bool_if:NT \l_@@_width_used_bool
1801 {
1802   \int_compare:nNnT \g_@@_total_X_weight_int = 0
1803     { \@@_error_or_warning:n { width-without-X-columns } }
1804 }
```

Now, if there is at least one X-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, \l\_@@\_X\_columns\_dim will be the width of a column of weight 1. For a X-column of weight  $n$ , the width will be \l\_@@\_X\_columns\_dim multiplied by  $n$ .

```
1805 \int_compare:nNnT \g_@@_total_X_weight_int > 0
1806 {
1807   \tl_gput_right:Nx \g_@@_aux_tl
1808     {
1809       \bool_set_true:N \l_@@_X_columns_aux_bool
1810       \dim_set:Nn \l_@@_X_columns_dim
1811         {
1812           \dim_compare:nNnTF
1813             {
1814               \dim_abs:n
1815                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1816             }
1817             <
1818             { 0.001 pt }
1819             { \dim_use:N \l_@@_X_columns_dim }
1820             {
1821               \dim_eval:n
1822                 {
1823                   ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1824                   / \int_use:N \g_@@_total_X_weight_int
1825                   + \l_@@_X_columns_dim
1826                 }
1827             }
1828         }
1829     }
1830 }
```

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 actual number of rows of the array).

```
1831 \int_compare:nNnT \l_@@_last_row_int > { -2 }
```

```

1832 {
1833   \bool_if:NF \l_@@_last_row_without_value_bool
1834   {
1835     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1836     {
1837       \@@_error:n { Wrong~last~row }
1838       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1839     }
1840   }
1841 }

```

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>73</sup>

```

1842 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1843 \bool_if:nTF \g_@@_last_col_found_bool
1844 { \int_gdecr:N \c@jCol }
1845 {
1846   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1847   { \@@_error:n { last~col~not~used } }
1848 }

```

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

```

1849 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1850 \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. 142).

```

1851 \int_compare:nNnT \l_@@_first_col_int = 0
1852 {
1853   \skip_horizontal:N \col@sep
1854   \skip_horizontal:N \g_@@_width_first_col_dim
1855 }

```

The construction of the real box is different when `\g_@@_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.

```

1856 \bool_if:NTF \g_@@_NiceArray_bool
1857 {
1858   \str_case:VnF \l_@@_baseline_tl
1859   {
1860     b \@@_use_arraybox_with_notes_b:
1861     c \@@_use_arraybox_with_notes_c:
1862   }
1863   \@@_use_arraybox_with_notes:
1864 }

```

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).

```

1865 {
1866   \int_compare:nNnTF \l_@@_first_row_int = 0
1867   {
1868     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1869     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1870   }
1871   { \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>74</sup>

```

1872 \int_compare:nNnTF \l_@@_last_row_int > { -2 }

```

<sup>73</sup>We remind that the potential “first column” (exterior) has the number 0.

<sup>74</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).

```

1873     {
1874         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1875         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1876     }
1877     { \dim_zero:N \l_tmpb_dim }
1878     \hbox_set:Nn \l_tmpa_box
1879     {
1880         \c_math_toggle_token
1881         \@@_color:V \l_@@_delimiters_color_tl
1882         \exp_after:wN \left \g_@@_left_delim_tl
1883         \vcenter
1884         {

```

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.

```

1885         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1886         \hbox
1887         {
1888             \bool_if:NTF \l_@@_NiceTabular_bool
1889             { \skip_horizontal:N -\tabcolsep }
1890             { \skip_horizontal:N -\arraycolsep }
1891             \@@_use_arraybox_with_notes_c:
1892             \bool_if:NTF \l_@@_NiceTabular_bool
1893             { \skip_horizontal:N -\tabcolsep }
1894             { \skip_horizontal:N -\arraycolsep }
1895         }

```

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

```

1896         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1897     }

```

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.

```

1898         \@@_color:V \l_@@_delimiters_color_tl
1899         \exp_after:wN \right \g_@@_right_delim_tl
1900         \c_math_toggle_token
1901     }

```

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.

```

1902     \bool_if:NTF \l_@@_delimiters_max_width_bool
1903     {
1904         \@@_put_box_in_flow_bis:nn
1905         \g_@@_left_delim_tl \g_@@_right_delim_tl
1906     }
1907     \@@_put_box_in_flow:
1908 }

```

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. 143).

```

1909     \bool_if:NT \g_@@_last_col_found_bool
1910     {
1911         \skip_horizontal:N \g_@@_width_last_col_dim
1912         \skip_horizontal:N \col@sep
1913     }
1914     \bool_if:NF \l_@@_Matrix_bool
1915     {
1916         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1917         { \@@_warning_gredirect_none:n { columns-not-used } }
1918     }
1919     \@@_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 exponent to a matrix in a mathematical formula.

```

1920     \egroup

```



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

```

1921 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1922 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1923 \iow_now:Nx \@mainaux
1924 {
1925   \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ t1 }
1926   { \exp_not:V \g_@@_aux_t1 }
1927 }
1928 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1929 \bool_if:NT \c_@@_footnote_bool \endsavenotes
1930 }

```

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.<sup>75</sup>

The preamble given by the final user is in `\g_@@_preamble_t1` and the modified version will be stored in `\g_@@_preamble_t1` also.

```

1931 \cs_new_protected:Npn \@@_transform_preamble:
1932 {

```

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 all column types (defined by the user or by various packages using `\newcolumnntype`).

Since we use the tools of `array` to do this expansion, we will have a programming which is not in the style of the L3 programming layer.

We redefine the column types `w` and `W`. We use `\@@_newcolumnntype` instead of `\newcolumnntype` because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters `w` and `W`. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able to use the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```

1933 \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).

```

1934 \bool_if:NF \l_@@_Matrix_bool
1935 {
1936   \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1937   \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

If the package `varwidth` has defined the column type `V`, we protect from expansion by redefining it to `\@@_V:` (which will be caught by our system).

```

1938 \cs_if_exist:NT \NC@find@V { \@@_newcolumnntype V { \@@_V: } }

```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by the L3 programming layer).

```

1939 \exp_args:NV \@temptokena \g_@@_preamble_t1

```

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

```

1940 \@tempwatrue

```

---

<sup>75</sup>Be careful: the transformation of the preamble may also have by-side effects, for example, the boolean `\g_@@_NiceArray_bool` will be set to `false` if we detect in the preamble a delimiter at the beginning or at the end.

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

```
1941 \@whilesw \if@tempswa \fi { \@tempwafalse \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`.

```
1942 \int_gzero:N \c@jCol
1943 \tl_gclear:N \g_@@_preamble_tl
```

`\g_tmpb_bool` will be raised if you have a `|` at the end of the preamble.

```
1944 \bool_gset_false:N \g_tmpb_bool
1945 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1946 {
1947   \tl_gset:Nn \g_@@_preamble_tl
1948     { ! { \skip_horizontal:N \arrayrulewidth } }
1949 }
1950 {
1951   \clist_if_in:NnT \l_@@_vlines_clist 1
1952   {
1953     \tl_gset:Nn \g_@@_preamble_tl
1954       { ! { \skip_horizontal:N \arrayrulewidth } }
1955   }
1956 }
```

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

```
1957 \seq_clear:N \g_@@_cols_vlism_seq
```

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

```
1958 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
1959 \int_zero:N \l_tmpa_int
```

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

```
1960 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1961 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1962 }
```

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

```
1963 \bool_if:NT \l_@@_colortbl_like_bool
1964 {
1965   \regex_replace_all:NnN
1966     \c_@@_columncolor_regex
1967     { \c { @@_columncolor_preamble } }
1968     \g_@@_preamble_tl
1969 }
```

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

```
1970 \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}`.

```
1971 \bool_lazy_or:nnT
1972   { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1973   { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1974   { \bool_gset_false:N \g_@@_NiceArray_bool }
```

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

```
1975 \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }
```

We complete the preamble with the potential “exterior columns” (on both sides).

```
1976 \int_compare:nNnTF \l_@@_first_col_int = 0
1977 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1978 {
1979   \bool_lazy_all:nT
1980   {
1981     \g_@@_NiceArray_bool
1982     { \bool_not_p:n \l_@@_NiceTabular_bool }
1983     { \tl_if_empty_p:N \l_@@_vlines_clist }
1984     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1985   }
1986   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1987 }
1988 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1989 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1990 {
1991   \bool_lazy_all:nT
1992   {
1993     \g_@@_NiceArray_bool
1994     { \bool_not_p:n \l_@@_NiceTabular_bool }
1995     { \tl_if_empty_p:N \l_@@_vlines_clist }
1996     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1997   }
1998   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1999 }
```

We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it’s not possible to do that in {NiceTabular\*} (we control that with the value of \l\_@@\_tabular\_width\_dim).

```
2000 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2001 {
2002   \tl_gput_right:Nn \g_@@_preamble_tl
2003   { > { \@@_error_too_much_cols: } 1 }
2004 }
2005 }
```

The command \@@\_patch\_preamble:n is the main function for the transformation of the preamble. It is recursive.

```
2006 \cs_new_protected:Npn \@@_patch_preamble:n #1
2007 {
2008   \str_case:nnF { #1 }
2009   {
2010     c { \@@_patch_preamble_i:n #1 }
2011     l { \@@_patch_preamble_i:n #1 }
2012     r { \@@_patch_preamble_i:n #1 }
2013     > { \@@_patch_preamble_xiv:n }
2014     ! { \@@_patch_preamble_ii:nn #1 }
2015     @ { \@@_patch_preamble_ii:nn #1 }
2016     | { \@@_patch_preamble_iii:n #1 }
2017     p { \@@_patch_preamble_iv:n #1 }
2018     b { \@@_patch_preamble_iv:n #1 }
2019     m { \@@_patch_preamble_iv:n #1 }
2020     \@@_V: { \@@_patch_preamble_v:n }
2021     V { \@@_patch_preamble_v:n }
2022     \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
2023     \@@_W: { \@@_patch_preamble_vi:nnnn { \@@_special_W: } #1 }
2024     \@@_S: { \@@_patch_preamble_vii:n }
2025     ( { \@@_patch_preamble_viii:nn #1 }
2026     [ { \@@_patch_preamble_viii:nn #1 }
```

```

2027   \{ { \@@_patch_preamble_viii:nn #1 }
2028   ) { \@@_patch_preamble_ix:nn #1 }
2029   ] { \@@_patch_preamble_ix:nn #1 }
2030   \} { \@@_patch_preamble_ix:nn #1 }
2031   X { \@@_patch_preamble_x:n }

```

When `tabularx` is loaded, a local redefinition of the specifier `X` is done to replace `X` by `\@@_X`. Thus, our column type `X` will be used in the `{NiceTabularX}`.

```

2032   \@@_X { \@@_patch_preamble_x:n }
2033   \q_stop { }
2034   }
2035   {
2036   \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
2037   {
2038     \seq_gput_right:Nx \g_@@_cols_vlism_seq
2039     { \int_eval:n { \c@jCol + 1 } }
2040     \tl_gput_right:Nx \g_@@_preamble_tl
2041     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2042     \@@_patch_preamble:n
2043   }

```

Now the case of a letter set by the final user for a customized rule. Such customized rule is defined by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs. Among the keys available in that list, there is the key `letter`. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix/ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```

2044   {
2045     \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2046     {
2047       \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2048       \@@_patch_preamble:n
2049     }
2050     { \@@_fatal:nn { unknown-column-type } { #1 } }
2051   }
2052 }
2053 }

```

Now, we will list all the auxiliary functions for the different types of entries in the preamble of the array.

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

```

2054 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
2055 {
2056   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2057   \tl_gclear:N \g_@@_pre_cell_tl
2058   \tl_gput_right:Nn \g_@@_preamble_tl
2059   {
2060     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2061     #1
2062     < \@@_cell_end:
2063   }

```

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

```

2064   \int_gincr:N \c@jCol
2065   \@@_patch_preamble_xi:n
2066 }

```

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

```

2067 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
2068 {
2069   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2070   \@@_patch_preamble:n
2071 }

```

For |

```
2072 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
2073 {
\l_tmpa_int is the number of successive occurrences of |
2074   \int_incr:N \l_tmpa_int
2075   \@@_patch_preamble_iii_i:n
2076 }
2077 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
2078 {
2079   \str_if_eq:nnTF { #1 } |
2080   { \@@_patch_preamble_iii:n | }
2081   {
2082     \dim_set:Nn \l_tmpa_dim
2083     {
2084       \arrayrulewidth * \l_tmpa_int
2085       + \doublerulesep * ( \l_tmpa_int - 1)
2086     }
2087     \tl_gput_right:Nx \g_@@_preamble_tl
2088     {
```

Here, the command `\dim_eval:n` is mandatory.

```
2089       \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_tmpa_dim } } }
2090     }
2091     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2092     {
2093       \@@_vline:n
2094       {
2095         position = \int_eval:n { \c@jCol + 1 } ,
2096         multiplicity = \int_use:N \l_tmpa_int ,
2097         total-width = \dim_use:N \l_tmpa_dim % added 2022-08-06
2098       }

```

We don't have provided value for `start` nor for `end`, which means that the rule will cover (potentially) all the rows of the array.

```
2099     }
2100     \int_zero:N \l_tmpa_int
2101     \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2102     \@@_patch_preamble:n #1
2103   }
2104 }
2105 \cs_new_protected:Npn \@@_patch_preamble_xiv:n #1
2106 {
2107   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #1 } }
2108   \@@_patch_preamble:n
2109 }
2110 \bool_new:N \l_@@_bar_at_end_of_pream_bool
```

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

```
2111 \keys_define:nn { WithArrows / p-column }
2112 {
2113   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2114   r .value_forbidden:n = true ,
2115   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2116   c .value_forbidden:n = true ,
2117   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2118   l .value_forbidden:n = true ,
2119   R .code:n =
2120   \IfPackageLoadedTF { ragged2e }
2121   { \str_set:Nn \l_@@_hpos_col_str { R } }
2122   {
```

```

2123     \@@_error_or_warning:n { ragged2e-not-loaded }
2124     \str_set:Nn \l_@@_hpos_col_str { r }
2125   } ,
2126   R .value_forbidden:n = true ,
2127   L .code:n =
2128     \IfPackageLoadedTF { ragged2e }
2129     { \str_set:Nn \l_@@_hpos_col_str { L } }
2130     {
2131       \@@_error_or_warning:n { ragged2e-not-loaded }
2132       \str_set:Nn \l_@@_hpos_col_str { l }
2133     } ,
2134   L .value_forbidden:n = true ,
2135   C .code:n =
2136     \IfPackageLoadedTF { ragged2e }
2137     { \str_set:Nn \l_@@_hpos_col_str { C } }
2138     {
2139       \@@_error_or_warning:n { ragged2e-not-loaded }
2140       \str_set:Nn \l_@@_hpos_col_str { c }
2141     } ,
2142   C .value_forbidden:n = true ,
2143   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2144   S .value_forbidden:n = true ,
2145   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2146   p .value_forbidden:n = true ,
2147   t .meta:n = p ,
2148   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2149   m .value_forbidden:n = true ,
2150   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2151   b .value_forbidden:n = true ,
2152 }

```

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

```

2153 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
2154 {
2155   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2156   \@@_patch_preamble_iv_i:n
2157 }

2158 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
2159 {
2160   \str_if_eq:nnTF { #1 } { [ ]
2161     { \@@_patch_preamble_iv_ii:w [ ]
2162       { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
2163     }
2164   \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
2165     { \@@_patch_preamble_iv_iii:nn { #1 } }

```

#1 is the optional argument of the specifier (a list of *key-value* pairs).

#2 is the mandatory argument of the specifier: the width of the column.

```

2166 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
2167 {

```

The possible values of \l\_@@\_hpos\_col\_str are j (for *justified* which is the initial value), l, c, r, L, C and R (when the user has used the corresponding key in the optional argument of the specifier).

```

2168   \str_set:Nn \l_@@_hpos_col_str { j }
2169   \tl_set:Nn \l_tmpa_tl { #1 }
2170   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2171   \@@_keys_p_column:V \l_tmpa_tl
2172   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2173 }

```

```

2174 \cs_new_protected:Npn \@@_keys_p_column:n #1
2175   { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2176 \cs_generate_variant:Nn \@@_keys_p_column:n { V }

```

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

```

2177 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
2178   {
2179     \use:x
2180     {
2181       \@@_patch_preamble_iv_v:nnnnnnn
2182       { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2183       { \dim_eval:n { #1 } }
2184     }

```

The parameter `\l_@@_hpos_col_str` (as `\l_@@_vpos_col_str`) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter `\l_@@_hpos_cell_str` which will provide the horizontal alignment of the column to which belongs the cell.

```

2185     \str_if_eq:VnTF \l_@@_hpos_col_str j
2186     { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
2187     {
2188       \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2189       { \str_lowercase:V \l_@@_hpos_col_str }
2190     }
2191     \str_case:Vn \l_@@_hpos_col_str
2192     {
2193       c { \exp_not:N \centering }
2194       l { \exp_not:N \raggedright }
2195       r { \exp_not:N \raggedleft }
2196       C { \exp_not:N \Centering }
2197       L { \exp_not:N \RaggedRight }
2198       R { \exp_not:N \RaggedLeft }
2199     }
2200   }
2201   { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2202   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2203   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2204   { #2 }
2205   {
2206     \str_case:VnF \l_@@_hpos_col_str
2207     {
2208       { j } { c }
2209       { si } { c }
2210     }

```

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

```

2211     { \str_lowercase:V \l_@@_hpos_col_str }
2212   }
2213 }

```

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

```

2214   \int_gincr:N \c@jCol
2215   \@@_patch_preamble_xi:n
2216 }

```

**#1** is the optional argument of `{minipage}` (or `{varwidth}`): `t` of `b`. Indeed, for the columns of type `m`, we use the value `b` here because there is a special post-action in order to center vertically the box (see **#4**).

**#2** is the width of the `{minipage}` (or `{varwidth}`), that is to say also the width of the column.

**#3** is the coding for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing). It's also possible to put in that **#3** some code to fix the value of `\l_@@_hpos_cell_str` which will be available in each cell of the column.

#4 is an extra-code which contains `\@@_center_cell_box:` (when the column is a m column) or nothing (in the other cases).

#5 is a code put just before the c (or r or l: see #8).

#6 is a code put just after the c (or r or l: see #8).

#7 is the type of environment: `minipage` or `varwidth`.

#8 is the letter c or r or l which is the basic specifier of column which is used *in fine*.

```

2217 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2218   {
2219     \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2220     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2221     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty: } } }
2222     \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2223     \tl_gclear:N \g_@@_pre_cell_tl
2224     \tl_gput_right:Nn \g_@@_preamble_tl
2225     {
2226       > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

```

2227     \dim_set:Nn \l_@@_col_width_dim { #2 }
2228     \@@_cell_begin:w
2229     \begin { #7 } [ #1 ] { #2 }

```

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

```

2230     \everypar
2231     {
2232       \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2233       \everypar { }
2234     }

```

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

```

2235     #3

```

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

```

2236     \g_@@_row_style_tl
2237     \arraybackslash
2238     #5
2239   }
2240   #8
2241   < {
2242     #6

```

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

```

2243     \@finalstrut \@arstrutbox
2244     % \bool_if:NT \g_@@_rotate_bool { \raggedright \hspace = 3 cm }
2245     \end { #7 }

```

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

```

2246     #4
2247     \@@_cell_end:
2248   }
2249 }
2250 }

```

```

2251 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2252 {
2253   \peek_meaning:NT \unskip
2254   {
2255     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2256     {
2257       \box_set_wd:Nn \l_@@_cell_box \c_zero_dim

```



We put the following code in order to have a column with the correct width even when all the cells of the column are empty.

```

2258         \skip_horizontal:N \l_@@_col_width_dim
2259     }
2260 }
2261 #1
2262 }
2263 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2264 {
2265     \peek_meaning:NT \_siunitx_table_skip:n
2266     {
2267         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2268         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2269     }
2270     #1
2271 }

```

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

```

2272 \cs_new_protected:Npn \@@_center_cell_box:
2273 {

```

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

```

2274     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2275     {
2276         \int_compare:nNnT
2277         { \box_ht:N \l_@@_cell_box }
2278         >

```

Previously, we had `\@arstrutbox` and not `\strutbox` in the following line but the code in `array` has changed in v 2.5g and we follow the change (see *array: Correctly identify single-line m-cells* in *LaTeX News 36*).

```

2279         { \box_ht:N \strutbox }
2280     {
2281         \hbox_set:Nn \l_@@_cell_box
2282         {
2283             \box_move_down:nn
2284             {
2285                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2286                 + \baselineskip ) / 2
2287             }
2288             { \box_use:N \l_@@_cell_box }
2289         }
2290     }
2291 }
2292 }

```

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

```

2293 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2294 {
2295     \str_if_eq:nnTF { #1 } { [ ] }
2296     { \@@_patch_preamble_v_i:w [ ] }
2297     { \@@_patch_preamble_v_i:w [ ] { #1 } }
2298 }
2299 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2300 { \@@_patch_preamble_v_ii:nn { #1 } }
2301 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2302 {

```

```

2303 \str_set:Nn \l_@@_vpos_col_str { p }
2304 \str_set:Nn \l_@@_hpos_col_str { j }
2305 \tl_set:Nn \l_tmpa_tl { #1 }
2306 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2307 \@@_keys_p_column:V \l_tmpa_tl
2308 \bool_if:NTF \c_@@_varwidth_loaded_bool
2309 { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2310 {
2311   \@@_error_or_warning:n { varwidth~not~loaded }
2312   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2313 }
2314 }

```

For w and W

```

2315 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2316 {
2317   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2318   \tl_gclear:N \g_@@_pre_cell_tl
2319   \tl_gput_right:Nn \g_@@_preamble_tl
2320   {
2321     > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

```

2322     \dim_set:Nn \l_@@_col_width_dim { #4 }
2323     \hbox_set:Nw \l_@@_cell_box
2324     \@@_cell_begin:w
2325     \str_set:Nn \l_@@_hpos_cell_str { #3 }
2326   }
2327   < {
2328     \@@_cell_end:
2329     \hbox_set_end:
2330     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2331     #1
2332     \@@_adjust_size_box:
2333     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2334   }
2335 }
2336 }

```

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

```

2337 \int_gincr:N \c@jCol
2338 \@@_patch_preamble_xi:n
2339 }

```

```

2340 \cs_new_protected:Npn \@@_special_W:
2341 {
2342   \dim_compare:nNnT
2343     { \box_wd:N \l_@@_cell_box }
2344     >
2345     \l_@@_col_width_dim
2346     { \@@_warning:n { W~warning } }
2347 }

```

For `\@@_S:`. If the user has used `S[...]`, `S` has been replaced by `\@@_S:` during the first expansion of the preamble (done with the tools of standard LaTeX and array).

```

2348 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2349 {
2350   \str_if_eq:nnTF { #1 } { [ ] }
2351   { \@@_patch_preamble_vii_i:w [ ] }
2352   { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2353 }

```

```

2354 \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2355   { \@@_patch_preamble_vii_ii:n { #1 } }
2356 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2357   {

```

We test whether the version of nicematrix is at least 3.0. We will change the programming of the test further with something like `\@ifpackagelater`.

```

2358   \cs_if_exist:NTF \siunitx_cell_begin:w
2359   {
2360     \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2361     \tl_gclear:N \g_@@_pre_cell_tl
2362     \tl_gput_right:Nn \g_@@_preamble_tl
2363     {
2364       > {
2365         \@@_cell_begin:w
2366         \keys_set:nn { siunitx } { #1 }
2367         \siunitx_cell_begin:w
2368       }
2369       c
2370       < { \siunitx_cell_end: \@@_cell_end: }
2371     }

```

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

```

2372     \int_gincr:N \c@jCol
2373     \@@_patch_preamble_xi:n
2374   }
2375   { \@@_fatal:n { Version-of-siunitx-too-old } }
2376 }

```

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

```

2377 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2378   {
2379     \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.

```

2380     \int_compare:nNnTF \c@jCol = \c_zero_int
2381     {
2382       \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2383       {

```

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

```

2384         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2385         \tl_gset:Nn \g_@@_right_delim_tl { . }
2386         \@@_patch_preamble:n #2
2387       }
2388       {
2389         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2390         \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2391       }
2392     }
2393     { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2394   }

```

```

2395 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2396   {
2397     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2398     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2399     \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
2400     {
2401       \@@_error:nn { delimiter-after-opening } { #2 }
2402       \@@_patch_preamble:n
2403     }
2404     { \@@_patch_preamble:n #2 }
2405   }

```

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 an 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}`).

```

2406 \cs_new_protected:Npn \@_patch_preamble_ix:n #1 #2
2407 {
2408   \bool_if:NT \l_@@_small_bool { \@_fatal:n { Delimiter-with-small } }
2409   \tl_if_in:nnTF { ) ] \} } { #2 }
2410   { \@_patch_preamble_ix_i:nnn #1 #2 }
2411   {
2412     \tl_if_eq:nnTF { \q_stop } { #2 }
2413     {
2414       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2415       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2416       {
2417         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2418         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2419         { \@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2420         \@_patch_preamble:n #2
2421       }
2422     }
2423     {
2424       \tl_if_in:nnT { ( [ \{ } { #2 }
2425       { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2426       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2427       { \@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2428       \@_patch_preamble:n #2
2429     }
2430   }
2431 }

2432 \cs_new_protected:Npn \@_patch_preamble_ix_i:nnn #1 #2 #3
2433 {
2434   \tl_if_eq:nnTF { \q_stop } { #3 }
2435   {
2436     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2437     {
2438       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2439       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2440       { \@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2441       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2442     }
2443     {
2444       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2445       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2446       { \@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2447       \@_error:nn { double-closing-delimiter } { #2 }
2448     }
2449   }
2450   {
2451     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2452     { \@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2453     \@_error:nn { double-closing-delimiter } { #2 }
2454     \@_patch_preamble:n #3
2455   }
2456 }

```

For the case of a letter `X`. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a `[` after the letter `X`.

```

2457 \cs_new_protected:Npn \@_patch_preamble_x:n #1
2458 {
2459   \str_if_eq:nnTF { #1 } { [ ]
2460   { \@_patch_preamble_x_i:w [ ]

```

```

2461     { \@@_patch_preamble_x_i:w [ ] #1 }
2462   }
2463 \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2464   { \@@_patch_preamble_x_ii:n { #1 } }

```

#1 is the optional argument of the X specifier (a list of *key-value* pairs).

The following set of keys is for the specifier X in the preamble of the array. Such specifier may have as keys all the keys of { *WithArrows* / p-column } but also a key as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter \l\_@@\_weight\_int).

```

2465 \keys_define:nn { WithArrows / X-column }
2466   { unknown .code:n = \int_set:Nn \l_@@_weight_int { \l_keys_key_str } }

```

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

```

2467 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2468   {

```

The possible values of \l\_@@\_hpos\_col\_str are j (for *justified* which is the initial value), l, c and r (when the user has used the corresponding key in the optional argument of the specifier X).

```

2469   \str_set:Nn \l_@@_hpos_col_str { j }

```

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

```

2470   \tl_set:Nn \l_@@_vpos_col_str { p }

```

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

```

2471   \int_zero_new:N \l_@@_weight_int
2472   \int_set:Nn \l_@@_weight_int { 1 }
2473   \tl_set:Nn \l_tmpa_tl { #1 }
2474   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2475   \@@_keys_p_column:V \l_tmpa_tl
2476   % \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
2477   \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2478   \int_compare:nNnT \l_@@_weight_int < 0
2479     {
2480       \@@_error_or_warning:n { negative-weight }
2481       \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2482     }
2483   \int_gadd:Nn \g_@@_total_X_weight_int \l_@@_weight_int

```

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

```

2484   \bool_if:NTF \l_@@_X_columns_aux_bool
2485     {
2486       \@@_patch_preamble_iv_iv:nn
2487       { \l_@@_weight_int \l_@@_X_columns_dim }
2488       { minipage }
2489     }
2490     {
2491       \tl_gput_right:Nn \g_@@_preamble_tl
2492       {
2493         > {
2494           \@@_cell_begin:w
2495           \bool_set_true:N \l_@@_X_column_bool

```

The following code will nullify the box of the cell.

```

2496       \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2497       { \hbox_set:Nn \l_@@_cell_box { } }

```

We put a `{minipage}` to give to the user the ability to put a command such as `\centering` in the `\RowStyle`.

```

2498         \begin { minipage } { 5 cm } \arraybackslash
2499         }
2500         c
2501         < {
2502             \end { minipage }
2503             \@@_cell_end:
2504         }
2505     }
2506     \int_gincr:N \c@jCol
2507     \@@_patch_preamble_xi:n
2508 }
2509 }
```

After a specifier of column, we have to test whether there is one or several `<{. .}` because, after those potential `<{. .}`, we have to insert `!\skip_horizontal:N ...` when the key `vlines` is used. In fact, we have also to test whether there is, after the `<{. .}`, a `@{. .}`.

```

2510 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2511 {
2512     \str_if_eq:nnTF { #1 } { < }
2513     \@@_patch_preamble_xiii:n
2514     {
2515         \str_if_eq:nnTF { #1 } { @ }
2516         \@@_patch_preamble_xv:n
2517         {
2518             \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2519             {
2520                 \tl_gput_right:Nn \g_@@_preamble_tl
2521                 { ! { \skip_horizontal:N \arrayrulewidth } }
2522             }
2523             {
2524                 \exp_args:NNx
2525                 \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2526                 {
2527                     \tl_gput_right:Nn \g_@@_preamble_tl
2528                     { ! { \skip_horizontal:N \arrayrulewidth } }
2529                 }
2530             }
2531             \@@_patch_preamble:n { #1 }
2532         }
2533     }
2534 }
2535 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2536 {
2537     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2538     \@@_patch_preamble_xi:n
2539 }
```

We have to catch a `@{. .}` after a specifier of column because, if we have to draw a vertical rule, we have to add in that `@{. .}` a `\hskip` corresponding to the width of the vertical rule.

```

2540 \cs_new_protected:Npn \@@_patch_preamble_xv:n #1
2541 {
2542     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2543     {
2544         \tl_gput_right:Nn \g_@@_preamble_tl
2545         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2546     }
2547     {
2548         \exp_args:NNx
2549         \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2550         {
```

```

2551         \tl_gput_right:Nn \g_@@_preamble_tl
2552         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2553     }
2554     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { #1 } } }
2555 }
2556 \@@_patch_preamble:n
2557 }

2558 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2559 {
2560     \group_begin:
2561     \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2562     \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
2563     \@temptokena { #2 }
2564     \@tempswatrue
2565     \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2566     \tl_gclear:N \g_@@_preamble_tl
2567     \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
2568     \group_end:
2569     \tl_set_eq:NN #1 \g_@@_preamble_tl
2570 }

```

## The redefinition of `\multicolumn`

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

```

2571 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2572 {

```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```

2573     \multispan { #1 }
2574     \begingroup
2575     \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
2576     \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2577     \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

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

```

2578     \@temptokena = { #2 }
2579     \@tempswatrue
2580     \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

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

```

2581     \tl_gclear:N \g_@@_preamble_tl
2582     \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

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

```

2583     \exp_args:NV \@mkpream \g_@@_preamble_tl
2584     \@addtopreamble \@empty
2585     \endgroup

```

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

```

2586     \int_compare:nNnT { #1 } > 1
2587     {
2588         \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2589         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2590         \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2591         \seq_gput_right:Nx \g_@@_pos_of_blocks_seq

```

```

2592     {
2593     {
2594         \int_compare:nNnTF \c@jCol = 0
2595         { \int_eval:n { \c@iRow + 1 } }
2596         { \int_use:N \c@iRow }
2597     }
2598     { \int_eval:n { \c@jCol + 1 } }
2599     {
2600         \int_compare:nNnTF \c@jCol = 0
2601         { \int_eval:n { \c@iRow + 1 } }
2602         { \int_use:N \c@iRow }
2603     }
2604     { \int_eval:n { \c@jCol + #1 } }
2605     { } % for the name of the block
2606 }
2607 }

```

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

```

2608     \cs_set:Npn \@sharp { #3 }
2609     \@arstrut
2610     \@preamble
2611     \null

```

We add some lines.

```

2612     \int_gadd:Nn \c@jCol { #1 - 1 }
2613     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2614     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2615     \ignorespaces
2616 }

```

The following commands will patch the (small) preamble of the `\multicolumn`. All those commands have a `m` in their name to recall that they deal with the redefinition of `\multicolumn`.

```

2617 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2618 {
2619     \str_case:nnF { #1 }
2620     {
2621         c { \@@_patch_m_preamble_i:n #1 }
2622         l { \@@_patch_m_preamble_i:n #1 }
2623         r { \@@_patch_m_preamble_i:n #1 }
2624         > { \@@_patch_m_preamble_ii:nn #1 }
2625         ! { \@@_patch_m_preamble_ii:nn #1 }
2626         @ { \@@_patch_m_preamble_ii:nn #1 }
2627         | { \@@_patch_m_preamble_iii:n #1 }
2628         p { \@@_patch_m_preamble_iv:nnn t #1 }
2629         m { \@@_patch_m_preamble_iv:nnn c #1 }
2630         b { \@@_patch_m_preamble_iv:nnn b #1 }
2631         \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2632         \@@_W: { \@@_patch_m_preamble_v:nnnn { \@@_special_W: } #1 }
2633         \q_stop { }
2634     }
2635     { \@@_fatal:nn { unknown~column~type } { #1 } }
2636 }

```

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

```

2637 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2638 {
2639     \tl_gput_right:Nn \g_@@_preamble_tl
2640     {
2641         > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2642         #1
2643         < \@@_cell_end:
2644     }

```



We test for the presence of a <.

```
2645 \@@_patch_m_preamble_x:n
2646 }
```

For >, ! and @

```
2647 \cs_new_protected:Npn \@@_patch_m_preamble_ii:n #1 #2
2648 {
2649 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2650 \@@_patch_m_preamble:n
2651 }
```

For |

```
2652 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2653 {
2654 \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2655 \@@_patch_m_preamble:n
2656 }
```

For p, m and b

```
2657 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2658 {
2659 \tl_gput_right:Nn \g_@@_preamble_tl
2660 {
2661 > {
2662 \@@_cell_begin:w
2663 \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2664 \mode_leave_vertical:
2665 \arraybackslash
2666 \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2667 }
2668 c
2669 < {
2670 \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2671 \end { minipage }
2672 \@@_cell_end:
2673 }
2674 }
```

We test for the presence of a <.

```
2675 \@@_patch_m_preamble_x:n
2676 }
```

For w and W

```
2677 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2678 {
2679 \tl_gput_right:Nn \g_@@_preamble_tl
2680 {
2681 > {
2682 \dim_set:Nn \l_@@_col_width_dim { #4 }
2683 \hbox_set:Nw \l_@@_cell_box
2684 \@@_cell_begin:w
2685 \str_set:Nn \l_@@_hpos_cell_str { #3 }
2686 }
2687 c
2688 < {
2689 \@@_cell_end:
2690 \hbox_set_end:
2691 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2692 #1
2693 \@@_adjust_size_box:
2694 \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2695 }
2696 }
```

We test for the presence of a <.

```
2697 \@@_patch_m_preamble_x:n
2698 }
```

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

```
2699 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2700 {
2701   \str_if_eq:nnTF { #1 } { < }
2702   \@@_patch_m_preamble_ix:n
2703   { \@@_patch_m_preamble:n { #1 } }
2704 }
2705 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2706 {
2707   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2708   \@@_patch_m_preamble_x:n
2709 }
```

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`).

```
2710 \cs_new_protected:Npn \@@_put_box_in_flow:
2711 {
2712   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2713   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2714   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2715   { \box_use_drop:N \l_tmpa_box }
2716   \@@_put_box_in_flow_i:
2717 }
```

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).

```
2718 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2719 {
2720   \pgfpicture
2721   \@@_qpoint:n { row - 1 }
2722   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2723   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2724   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2725   \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).

```
2726 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2727 {
2728   \int_set:Nn \l_tmpa_int
2729   {
2730     \str_range:Nnn
2731     \l_@@_baseline_tl
2732     6
2733     { \tl_count:V \l_@@_baseline_tl }
2734   }
2735   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2736 }
2737 {
2738   \str_case:VnF \l_@@_baseline_tl
2739   {
2740     { t } { \int_set:Nn \l_tmpa_int 1 }
2741     { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2742   }
2743   { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2744   \bool_lazy_or:nnT
```

```

2745     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2746     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2747     {
2748         \@@_error:n { bad~value~for~baseline }
2749         \int_set:Nn \l_tmpa_int 1
2750     }
2751     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2752     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2753     }
2754     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2755     \endpgfpicture
2756     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2757     \box_use_drop:N \l_tmpa_box
2758 }

```

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).

```

2759 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2760 {

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

2761     \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
2762     {
2763         \box_set_wd:Nn \l_@@_the_array_box
2764         { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2765     }

```

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

```

2766     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2767     \bool_if:NT \l_@@_caption_above_bool
2768     {
2769         \tl_if_empty:NF \l_@@_caption_tl
2770         {
2771             \bool_set_false:N \g_@@_caption_finished_bool
2772             \int_gzero:N \c@tabularnote
2773             \@@_insert_caption:

```

If there is one or several commands `\tabularnote` in the caption, we will write in the `aux` file the number of such tabular notes.

```

2774         \int_gset:Nn \c@tabularnote
2775         { \seq_count:N \g_@@_notes_in_caption_seq }
2776         \int_compare:nNnF \c@tabularnote = 0
2777         {
2778             \tl_gput_right:Nx \g_@@_aux_tl
2779             {
2780                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2781                 { \int_eval:n { \c@tabularnote } }
2782             }
2783         }
2784     }
2785 }

```

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

```

2786     \hbox
2787     {
2788         \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.

```

2789     \@@_create_extra_nodes:
2790     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2791 }

```

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

```

2792 \bool_lazy_any:nT
2793 {
2794   { ! \seq_if_empty_p:N \g_@@_notes_seq }
2795   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2796   { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2797 }
2798 \@@_insert_tabularnotes:
2799 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2800 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2801 \end { minipage }
2802 }

```

```

2803 \cs_new_protected:Npn \@@_insert_caption:
2804 {
2805   \tl_if_empty:NF \l_@@_caption_tl
2806   {
2807     \cs_if_exist:NTF \@capttype
2808     { \@@_insert_caption_i: }
2809     { \@@_error:n { caption-outside~float } }
2810   }
2811 }

```

```

2812 \cs_new_protected:Npn \@@_insert_caption_i:
2813 {
2814   \group_begin:

```

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

```

2815   \bool_set_true:N \l_@@_in_caption_bool

```

The package `floatrow` does a redefinition of `\@makecaption` which will extract the caption from the tabular. However, the old version of `\@makecaption` has been stored by `floatrow` in `\FR@makecaption`. That's why we restore the old version.

```

2816   \bool_if:NT \c_@@_floatrow_loaded_bool
2817   { \cs_set_eq:NN \@makecaption \FR@makecaption }
2818   \tl_if_empty:NTF \l_@@_short_caption_tl
2819   { \caption { \l_@@_caption_tl } }
2820   { \caption [ \l_@@_short_caption_tl ] { \l_@@_caption_tl } }
2821   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2822   \group_end:
2823 }

```

```

2824 \cs_new_protected:Npn \@@_tabularnote_error:n #1
2825 {
2826   \@@_error_or_warning:n { tabularnote~below~the~tabular }
2827   \@@_gredirect_none:n { tabularnote~below~the~tabular }
2828 }

```

```

2829 \cs_new_protected:Npn \@@_insert_tabularnotes:
2830 {
2831   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2832   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
2833   \skip_vertical:N 0.65ex

```

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

```

2834 \group_begin:
2835 \l_@@_notes_code_before_tl
2836 \tl_if_empty:NF \g_@@_tabularnote_tl
2837 {
2838   \g_@@_tabularnote_tl \par
2839   \tl_gclear:N \g_@@_tabularnote_tl
2840 }

```

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.

```

2841 \int_compare:nNnT \c@tabularnote > 0
2842 {
2843   \bool_if:NTF \l_@@_notes_para_bool
2844   {
2845     \begin { tabularnotes* }
2846     \seq_map_inline:Nn \g_@@_notes_seq { \item ##1 } \strut
2847     \end { tabularnotes* }

```

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

```

2848   \par
2849 }
2850 {
2851   \tabularnotes
2852   \seq_map_inline:Nn \g_@@_notes_seq { \item ##1 } \strut
2853   \endtabularnotes
2854 }
2855 }
2856 \unskip
2857 \group_end:
2858 \bool_if:NT \l_@@_notes_bottomrule_bool
2859 {
2860   \bool_if:NTF \c_@@_booktabs_loaded_bool
2861   {

```

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

```

2862   \skip_vertical:N \aboverulesep

```

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

```

2863   { \CT@arc@ \hrule height \heavyrulewidth }
2864 }
2865 { \@@_error_or_warning:n { bottomrule-without-booktabs } }
2866 }
2867 \l_@@_notes_code_after_tl
2868 \seq_gclear:N \g_@@_notes_seq
2869 \seq_gclear:N \g_@@_notes_in_caption_seq
2870 \int_gzero:N \c@tabularnote
2871 }

```

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`.

```

2872 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2873 {
2874   \pgfpicture
2875   \@@_qpoint:n { row - 1 }
2876   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2877   \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2878   \dim_gsub:Nn \g_tmpa_dim \pgf@y
2879   \endpgfpicture
2880   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2881   \int_compare:nNnT \l_@@_first_row_int = 0
2882   {

```

```

2883     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2884     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2885   }
2886   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2887 }

```

Now, the general case.

```

2888 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2889 {

```

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

```

2890   \tl_if_eq:NnT \l_@@_baseline_tl { t }
2891   { \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`.

```

2892   \pgfpicture
2893   \@@_qpoint:n { row - 1 }
2894   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2895   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2896   {
2897     \int_set:Nn \l_tmpa_int
2898     {
2899       \str_range:Nnn
2900         \l_@@_baseline_tl
2901         6
2902         { \tl_count:V \l_@@_baseline_tl }
2903     }
2904     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2905   }
2906   {
2907     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2908     \bool_lazy_or:nnT
2909     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2910     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2911     {
2912       \@@_error:n { bad-value-for-baseline }
2913       \int_set:Nn \l_tmpa_int 1
2914     }
2915     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2916   }
2917   \dim_gsub:Nn \g_tmpa_dim \pgf@y
2918   \endpgfpicture
2919   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2920   \int_compare:nNnT \l_@@_first_row_int = 0
2921   {
2922     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2923     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2924   }
2925   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2926 }

```

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.

```

2927 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2928 {

```

We will compute the real width of both delimiters used.

```

2929   \dim_zero_new:N \l_@@_real_left_delim_dim
2930   \dim_zero_new:N \l_@@_real_right_delim_dim
2931   \hbox_set:Nn \l_tmpb_box
2932   {
2933     \c_math_toggle_token

```

```

2934     \left #1
2935     \vcenter
2936     {
2937         \vbox_to_ht:nn
2938         { \box_ht_plus_dp:N \l_tmpa_box }
2939         { }
2940     }
2941     \right .
2942     \c_math_toggle_token
2943 }
2944 \dim_set:Nn \l_@@_real_left_delim_dim
2945 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2946 \hbox_set:Nn \l_tmpb_box
2947 {
2948     \c_math_toggle_token
2949     \left .
2950     \vbox_to_ht:nn
2951     { \box_ht_plus_dp:N \l_tmpa_box }
2952     { }
2953     \right #2
2954     \c_math_toggle_token
2955 }
2956 \dim_set:Nn \l_@@_real_right_delim_dim
2957 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2958     \skip_horizontal:N \l_@@_left_delim_dim
2959     \skip_horizontal:N -\l_@@_real_left_delim_dim
2960     \@@_put_box_in_flow:
2961     \skip_horizontal:N \l_@@_right_delim_dim
2962     \skip_horizontal:N -\l_@@_real_right_delim_dim
2963 }

```

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).

```

2964 \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).

```

2965 {
2966     \peek_remove_spaces:n
2967     {
2968         \peek_meaning:NTF \end
2969         \@@_analyze_end:Nn
2970         {
2971             \@@_transform_preamble:

```

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

```

2972         \@@_array:V \g_@@_preamble_tl
2973     }
2974 }
2975 }
2976 {
2977     \@@_create_col_nodes:
2978     \endarray
2979 }

```

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

```
2980 \NewDocumentEnvironment { @@-light-syntax } { b }
2981   {
```

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`.

```
2982   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2983   \tl_map_inline:nn { #1 }
2984     {
2985       \str_if_eq:nnT { ##1 } { & }
2986       { \@@_fatal:n { ampersand~in~light-syntax } }
2987       \str_if_eq:nnT { ##1 } { \ }
2988       { \@@_fatal:n { double-backslash~in~light-syntax } }
2989     }
```

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`.

```
2990   \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
2991   }
```

Now, the second part of the environment. We must leave these lines in the second part (and not put them in the first part even though we caught the whole body of the environment with an argument of type `b`) in order to have the columns `S` of `siunitx` working fine.

```
2992   {
2993     \@@_create_col_nodes:
2994     \endarray
2995   }
2996   \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
2997   {
2998     \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).

```
2999   \seq_clear_new:N \l_@@_rows_seq
```

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

```
3000   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3001   \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3002   \seq_pop_right:MN \l_@@_rows_seq \l_tmpa_tl
3003   \tl_if_empty:NF \l_tmpa_tl
3004     { \seq_put_right:NV \l_@@_rows_seq \l_tmpa_tl }
```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to compute that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```
3005   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3006     { \int_set:Nn \l_@@_last_row_int { \seq_count:N \l_@@_rows_seq } }
```

The new value of the body (that is to say after replacement of the separators of rows and columns by `\` and `&`) of the environment will be stored in `\l_@@_new_body_tl` (that part of the implementation has been changed in the version 6.11 of `nicematrix` in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`).

```
3007   \tl_clear_new:N \l_@@_new_body_tl
3008   \int_zero_new:N \l_@@_nb_cols_int
```



First, we treat the first row.

```
3009 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3010 \@@_line_with_light_syntax:V \l_tmpa_tl
```

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

```
3011 \seq_map_inline:Nn \l_@@_rows_seq
3012 {
3013   \tl_put_right:Nn \l_@@_new_body_tl { \ }
3014   \@@_line_with_light_syntax:n { ##1 }
3015 }
3016 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3017 {
3018   \int_set:Nn \l_@@_last_col_int
3019   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3020 }
```

Now, we can construct the preamble: if the user has used the key `last-col`, we have the correct number of columns even though the user has used `last-col` without value.

```
3021 \@@_transform_preamble:
```

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

```
3022 \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
3023 }
3024 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3025 {
3026   \seq_clear_new:N \l_@@_cells_seq
3027   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3028   \int_set:Nn \l_@@_nb_cols_int
3029   {
3030     \int_max:nn
3031     \l_@@_nb_cols_int
3032     { \seq_count:N \l_@@_cells_seq }
3033   }
3034   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3035   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
3036   \seq_map_inline:Nn \l_@@_cells_seq
3037   { \tl_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3038 }
3039 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { V }
```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security). When this command is used, `#1` is, in fact, always `\end`.

```
3040 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3041 {
3042   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3043   { \@@_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.

```
3044 \end { #2 }
3045 }
```

The command `\@@_create_col_nodes:` will construct a special last row. That last row is a false row used to create the `col` nodes and to fix the width of the columns (when the array is constructed with an option which specifies the width of the columns).

```
3046 \cs_new:Npn \@@_create_col_nodes:
3047 {
3048   \crcr
3049   \int_compare:nNnT \l_@@_first_col_int = 0
3050   {
```

```

3051 \omit
3052 \hbox_overlap_left:n
3053 {
3054   \bool_if:NT \l_@@_code_before_bool
3055     { \pgfsys@markposition { \@@_env: - col - 0 } }
3056   \pgfpicture
3057   \pgfrememberpicturerepositiononpagetrue
3058   \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3059   \str_if_empty:NF \l_@@_name_str
3060     { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3061   \endpgfpicture
3062   \skip_horizontal:N 2\col@sep
3063   \skip_horizontal:N \g_@@_width_first_col_dim
3064 }
3065 &
3066 }
3067 \omit

```

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

```

3068 \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`).

```

3069 \int_compare:nNnTF \l_@@_first_col_int = 0
3070 {
3071   \bool_if:NT \l_@@_code_before_bool
3072     {
3073       \hbox
3074         {
3075           \skip_horizontal:N -0.5\arrayrulewidth
3076           \pgfsys@markposition { \@@_env: - col - 1 }
3077           \skip_horizontal:N 0.5\arrayrulewidth
3078         }
3079     }
3080   \pgfpicture
3081   \pgfrememberpicturerepositiononpagetrue
3082   \pgfcoordinate { \@@_env: - col - 1 }
3083     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3084   \str_if_empty:NF \l_@@_name_str
3085     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3086   \endpgfpicture
3087 }
3088 {
3089   \bool_if:NT \l_@@_code_before_bool
3090     {
3091       \hbox
3092         {
3093           \skip_horizontal:N 0.5\arrayrulewidth
3094           \pgfsys@markposition { \@@_env: - col - 1 }
3095           \skip_horizontal:N -0.5\arrayrulewidth
3096         }
3097     }
3098   \pgfpicture
3099   \pgfrememberpicturerepositiononpagetrue
3100   \pgfcoordinate { \@@_env: - col - 1 }
3101     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3102   \str_if_empty:NF \l_@@_name_str
3103     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3104   \endpgfpicture
3105 }

```

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.

```

3106   \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3107   \bool_if:NF \l_@@_auto_columns_width_bool
3108     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3109     {
3110       \bool_lazy_and:nnTF
3111         \l_@@_auto_columns_width_bool
3112         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3113         { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3114         { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3115       \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3116     }
3117   \skip_horizontal:N \g_tmpa_skip
3118   \hbox
3119     {
3120       \bool_if:NT \l_@@_code_before_bool
3121       {
3122         \hbox
3123           {
3124             \skip_horizontal:N -0.5\arrayrulewidth
3125             \pgfsys@markposition { \@@_env: - col - 2 }
3126             \skip_horizontal:N 0.5\arrayrulewidth
3127           }
3128         }
3129       \pgfpicture
3130       \pgfrememberpicturepositiononpagetrue
3131       \pgfcoordinate { \@@_env: - col - 2 }
3132         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3133       \str_if_empty:NF \l_@@_name_str
3134         { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3135       \endpgfpicture
3136     }

```

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.

```

3137   \int_gset:Nn \g_tmpa_int 1
3138   \bool_if:NTF \g_@@_last_col_found_bool
3139     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3140     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3141     {
3142       &
3143       \omit
3144       \int_gincr:N \g_tmpa_int

```

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

```

3145       \skip_horizontal:N \g_tmpa_skip
3146       \bool_if:NT \l_@@_code_before_bool
3147       {
3148         \hbox
3149           {
3150             \skip_horizontal:N -0.5\arrayrulewidth
3151             \pgfsys@markposition
3152               { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3153             \skip_horizontal:N 0.5\arrayrulewidth
3154           }
3155       }

```

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

```

3156       \pgfpicture
3157       \pgfrememberpicturepositiononpagetrue
3158       \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3159         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3160       \str_if_empty:NF \l_@@_name_str

```

```

3161     {
3162         \pgfnodealias
3163         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3164         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3165     }
3166 \endpgfpicture
3167 }

3168 &
3169 \omit

```

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

```

3170 \int_compare:nNnT \g_@@_col_total_int = 1
3171 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3172 \skip_horizontal:N \g_tmpa_skip
3173 \int_gincr:N \g_tmpa_int
3174 \bool_lazy_all:nT
3175 {
3176     \g_@@_NiceArray_bool
3177     { \bool_not_p:n \l_@@_NiceTabular_bool }
3178     { \clist_if_empty_p:N \l_@@_vlines_clist }
3179     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3180     { ! \l_@@_bar_at_end_of_pream_bool }
3181 }
3182 { \skip_horizontal:N -\col@sep }
3183 \bool_if:NT \l_@@_code_before_bool
3184 {
3185     \hbox
3186     {
3187         \skip_horizontal:N -0.5\arrayrulewidth

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

3188     \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3189     { \skip_horizontal:N -\arraycolsep }
3190 \pgfsys@markposition
3191 { \@@_env: - col - \int_eval:n {
3192     \g_tmpa_int + 1 } }
3193 \skip_horizontal:N 0.5\arrayrulewidth
3194 \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
3195     { \skip_horizontal:N \arraycolsep }
3196 }
3197 }
3198 \pgfpicture
3199 \pgfrememberpicturepositiononpagetrue
3200 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3201 {
3202     \bool_lazy_and:nnTF \l_@@_Matrix_bool \g_@@_NiceArray_bool
3203     {
3204         \pgfpoint
3205         { - 0.5 \arrayrulewidth - \arraycolsep }
3206         \c_zero_dim
3207     }
3208     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3209 }
3210 \str_if_empty:NF \l_@@_name_str
3211 {
3212     \pgfnodealias
3213     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3214     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3215 }
3216 \endpgfpicture

```

```

3217 \bool_if:NT \g_@@_last_col_found_bool
3218 {
3219   \hbox_overlap_right:n
3220   {
3221     \skip_horizontal:N \g_@@_width_last_col_dim
3222     \bool_if:NT \l_@@_code_before_bool
3223     {
3224       \pgfsys@markposition
3225       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3226     }
3227     \pgfpicture
3228     \pgfrememberpicturepositiononpagetrue
3229     \pgfcoordinate
3230     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3231     \pgfpointorigin
3232     \str_if_empty:NF \l_@@_name_str
3233     {
3234       \pgfnodealias
3235       {
3236         \l_@@_name_str - col
3237         - \int_eval:n { \g_@@_col_total_int + 1 }
3238       }
3239       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3240     }
3241     \endpgfpicture
3242   }
3243 }
3244 \cr
3245 }

```

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

```

3246 \tl_const:Nn \c_@@_preamble_first_col_tl
3247 {
3248   >
3249   {

```

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

```

3250   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3251   \bool_gset_true:N \g_@@_after_col_zero_bool
3252   \@@_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.

```

3253   \hbox_set:Nw \l_@@_cell_box
3254   \@@_math_toggle_token:
3255   \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”.

```

3256   \bool_lazy_and:nnT
3257   { \int_compare_p:nNn \c@iRow > 0 }
3258   {
3259     \bool_lazy_or_p:nn
3260     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3261     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3262   }
3263   {
3264     \l_@@_code_for_first_col_tl
3265     \xglobal \colorlet { nicematrix-first-col } { . }
3266   }
3267 }

```

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`.

```

3268   l
3269   <
3270   {
3271     \@@_math_toggle_token:
3272     \hbox_set_end:
3273     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3274     \@@_adjust_size_box:
3275     \@@_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.

```

3276     \dim_gset:Nn \g_@@_width_first_col_dim
3277     { \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.

```

3278     \hbox_overlap_left:n
3279     {
3280       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3281       \@@_node_for_cell:
3282       { \box_use_drop:N \l_@@_cell_box }
3283       \skip_horizontal:N \l_@@_left_delim_dim
3284       \skip_horizontal:N \l_@@_left_margin_dim
3285       \skip_horizontal:N \l_@@_extra_left_margin_dim
3286     }
3287     \bool_gset_false:N \g_@@_empty_cell_bool
3288     \skip_horizontal:N -2\col@sep
3289   }
3290 }

```

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

```

3291 \tl_const:Nn \c_@@_preamble_last_col_tl
3292 {
3293   >
3294   {

```

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

```

3295     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

3296     \bool_gset_true:N \g_@@_last_col_found_bool
3297     \int_gincr:N \c@jCol
3298     \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.

```

3299     \hbox_set:Nw \l_@@_cell_box
3300     \@@_math_toggle_token:
3301     \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”.

```

3302     \int_compare:nNnT \c@iRow > 0
3303     {
3304       \bool_lazy_or:nnT
3305       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3306       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3307       {
3308         \l_@@_code_for_last_col_tl
3309         \xglobal \colorlet { nicematrix-last-col } { . }
3310       }
3311     }
3312   }
3313   l

```

```

3314 <
3315 {
3316   \@@_math_toggle_token:
3317   \hbox_set_end:
3318   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3319   \@@_adjust_size_box:
3320   \@@_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.

```

3321   \dim_gset:Nn \g_@@_width_last_col_dim
3322   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3323   \skip_horizontal:N -2\col@sep

```

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

```

3324   \hbox_overlap_right:n
3325   {
3326     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3327     {
3328       \skip_horizontal:N \l_@@_right_delim_dim
3329       \skip_horizontal:N \l_@@_right_margin_dim
3330       \skip_horizontal:N \l_@@_extra_right_margin_dim
3331       \@@_node_for_cell:
3332     }
3333   }
3334   \bool_gset_false:N \g_@@_empty_cell_bool
3335 }
3336 }

```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}` but, in fact, there is a flag `\g_@@_NiceArray_bool`. In `{NiceArrayWithDelims}`, some special code will be executed if this flag is raised.

```

3337 \NewDocumentEnvironment { NiceArray } { }
3338 {
3339   \bool_gset_true:N \g_@@_NiceArray_bool
3340   \str_if_empty:NT \g_@@_name_env_str
3341   { \str_gset:Nn \g_@@_name_env_str { NiceArray } }

```

We put `.` and `.` for the delimiters but, in fact, that doesn’t matter because these arguments won’t be used in `{NiceArrayWithDelims}` (because the flag `\g_@@_NiceArray_bool` is raised).

```

3342   \NiceArrayWithDelims . .
3343 }
3344 { \endNiceArrayWithDelims }

```

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

```

3345 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3346 {
3347   \NewDocumentEnvironment { #1 NiceArray } { }
3348   {
3349     \bool_gset_false:N \g_@@_NiceArray_bool
3350     \str_if_empty:NT \g_@@_name_env_str
3351     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3352     \@@_test_if_math_mode:
3353     \NiceArrayWithDelims #2 #3
3354   }
3355   { \endNiceArrayWithDelims }
3356 }
3357 \@@_def_env:nnn p ( )
3358 \@@_def_env:nnn b [ ]
3359 \@@_def_env:nnn B \{ \}
3360 \@@_def_env:nnn v | |
3361 \@@_def_env:nnn V \| \|

```

## The environment `{NiceMatrix}` and its variants

```

3362 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3363 {
3364   \bool_set_true:N \l_@@_Matrix_bool
3365   \use:c { #1 NiceArray }
3366   {
3367     *
3368     {
3369       \int_case:nnF \l_@@_last_col_int
3370       {
3371         { -2 } { \c@MaxMatrixCols }
3372         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3373       }
3374       { \int_eval:n { \l_@@_last_col_int - 1 } }
3375     }
3376     { #2 }
3377   }
3378 }
3379 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3380 \clist_map_inline:nn { p , b , B , v , V }
3381 {
3382   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3383   {
3384     \bool_gset_false:N \g_@@_NiceArray_bool
3385     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3386     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3387     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3388   }
3389   { \use:c { end #1 NiceArray } }
3390 }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

We define also an environment `{NiceMatrix}`

```

3391 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3392 {
3393   \bool_gset_false:N \g_@@_NiceArray_bool
3394   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3395   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3396   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3397 }
3398 { \endNiceArray }

```

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

```

3399 \cs_new_protected:Npn \@@_NotEmpty:
3400 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

## `{NiceTabular}`, `{NiceTabularX}` and `{NiceTabular*}`

```

3401 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3402 {

```

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

```

3403   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3404     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3405   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3406   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3407   \tl_if_empty:NF \l_@@_short_caption_tl
3408     {
3409       \tl_if_empty:NT \l_@@_caption_tl
3410       {

```



```

3411         \@@_error_or_warning:n { short-caption-without-caption }
3412         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3413     }
3414 }
3415 \tl_if_empty:NF \l_@@_label_tl
3416 {
3417     \tl_if_empty:NT \l_@@_caption_tl
3418     { \@@_error_or_warning:n { label-without-caption } }
3419 }
3420 \NewDocumentEnvironment { TabularNote } { b }
3421 {
3422     \bool_if:NTF \l_@@_in_code_after_bool
3423     { \@@_error_or_warning:n { TabularNote-in-CodeAfter } }
3424     {
3425         \tl_if_empty:NF \g_@@_tabularnote_tl
3426         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3427         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3428     }
3429 }
3430 { }
3431 \bool_set_true:N \l_@@_NiceTabular_bool
3432 \NiceArray { #2 }
3433 }
3434 { \endNiceArray }

3435 \cs_set_protected:Npn \@@_newcolumnntype #1
3436 {
3437     \cs_if_free:cT { NC @ find @ #1 }
3438     { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3439     \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
3440     \peek_meaning:NTF [
3441         { \newcol@ #1 }
3442         { \newcol@ #1 [ 0 ] }
3443     ]

3444 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3445 {

The following code prevents the expansion of the ‘X’ columns with the definition of that columns in
tabularx (this would result in an error in {NiceTabularX}).
3446     \bool_if:NT \c_@@_tabularx_loaded_bool { \newcolumnntype { X } { \@@_X } }
3447     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3448     \dim_zero_new:N \l_@@_width_dim
3449     \dim_set:Nn \l_@@_width_dim { #1 }
3450     \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3451     \bool_set_true:N \l_@@_NiceTabular_bool
3452     \NiceArray { #3 }
3453 }
3454 { \endNiceArray }

3455 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
3456 {
3457     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3458     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3459     \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3460     \bool_set_true:N \l_@@_NiceTabular_bool
3461     \NiceArray { #3 }
3462 }
3463 { \endNiceArray }

```

**After the construction of the array**

```

3464 \cs_new_protected:Npn \@@_after_array:
3465 {
3466   \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.

```

3467   \bool_if:NT \g_@@_last_col_found_bool
3468     { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }

```

If we are in an environment without preamble (like `{NiceMatrix}` or `{pNiceMatrix}`) and if the option `last-col` has been used without value we also fix the real value of `\l_@@_last_col_int`.

```

3469   \bool_if:NT \l_@@_last_col_without_value_bool
3470     { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }

```

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

```

3471   \bool_if:NT \l_@@_last_row_without_value_bool
3472     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3473   \tl_gput_right:Nx \g_@@_aux_tl
3474   {
3475     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3476     {
3477       \int_use:N \l_@@_first_row_int ,
3478       \int_use:N \c@iRow ,
3479       \int_use:N \g_@@_row_total_int ,
3480       \int_use:N \l_@@_first_col_int ,
3481       \int_use:N \c@jCol ,
3482       \int_use:N \g_@@_col_total_int
3483     }
3484   }

```

We write also the potential content of `\g_@@_pos_of_blocks_seq`. It will be used to recreate the blocks with a name in the `\CodeBefore` and also if the command `\rowcolors` is used with the key `respect-blocks`).

```

3485   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3486   {
3487     \tl_gput_right:Nx \g_@@_aux_tl
3488     {
3489       \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3490       { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3491     }
3492   }
3493   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3494   {
3495     \tl_gput_right:Nx \g_@@_aux_tl
3496     {
3497       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3498       { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3499       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3500       { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3501     }
3502   }

```

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

```

3503   \@@_create_diag_nodes:

```

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

```

3504   \pgfpicture
3505   \int_step_inline:nn \c@iRow
3506   {
3507     \pgfnodealias

```

```

3508     { \@@_env: - ##1 - last }
3509     { \@@_env: - ##1 - \int_use:N \c@jCol }
3510   }
3511   \int_step_inline:nn \c@jCol
3512   {
3513     \pgfnodealias
3514     { \@@_env: - last - ##1 }
3515     { \@@_env: - \int_use:N \c@iRow - ##1 }
3516   }
3517   \str_if_empty:NF \l_@@_name_str
3518   {
3519     \int_step_inline:nn \c@iRow
3520     {
3521       \pgfnodealias
3522       { \l_@@_name_str - ##1 - last }
3523       { \@@_env: - ##1 - \int_use:N \c@jCol }
3524     }
3525     \int_step_inline:nn \c@jCol
3526     {
3527       \pgfnodealias
3528       { \l_@@_name_str - last - ##1 }
3529       { \@@_env: - \int_use:N \c@iRow - ##1 }
3530     }
3531   }
3532   \endpgfpicture

```

By default, the diagonal lines will be parallelized<sup>76</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.

```

3533   \bool_if:NT \l_@@_parallelize_diags_bool
3534   {
3535     \int_gzero_new:N \g_@@_ddots_int
3536     \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.

```

3537     \dim_gzero_new:N \g_@@_delta_x_one_dim
3538     \dim_gzero_new:N \g_@@_delta_y_one_dim
3539     \dim_gzero_new:N \g_@@_delta_x_two_dim
3540     \dim_gzero_new:N \g_@@_delta_y_two_dim
3541   }
3542   \int_zero_new:N \l_@@_initial_i_int
3543   \int_zero_new:N \l_@@_initial_j_int
3544   \int_zero_new:N \l_@@_final_i_int
3545   \int_zero_new:N \l_@@_final_j_int
3546   \bool_set_false:N \l_@@_initial_open_bool
3547   \bool_set_false:N \l_@@_final_open_bool

```

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

```

3548   \bool_if:NT \l_@@_small_bool
3549   {
3550     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3551     \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_end_dim` correspond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.

```

3552     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3553     { 0.6 \l_@@_xdots_shorten_start_dim }

```

---

<sup>76</sup>It's possible to use the option `parallelize-diags` to disable this parallelization.

```

3554     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3555     { 0.6 \l_@@_xdots_shorten_end_dim }
3556 }

```

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

```

3557 \@@_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.

```

3558 \@@_compute_corners:

```

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

```

3559 \@@_adjust_pos_of_blocks_seq:
3560 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3561 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3562 \bool_if:NT \c_@@_tikz_loaded_bool
3563 {
3564     \tikzset
3565     {
3566         every~picture / .style =
3567         {
3568             overlay ,
3569             remember~picture ,
3570             name-prefix = \@@_env: -
3571         }
3572     }
3573 }
3574 \cs_set_eq:NN \ialign \@@_old_ialign:
3575 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3576 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3577 \cs_set_eq:NN \OverBrace \@@_OverBrace
3578 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3579 \cs_set_eq:NN \line \@@_line
3580 \g_@@_pre_code_after_tl
3581 \tl_gclear:N \g_@@_pre_code_after_tl

```

When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it’s possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That’s why we set `\Code-after` to be *no-op* now.

```

3582 \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).

```

3583 \seq_gclear:N \g_@@_submatrix_names_seq

```

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

```

3584 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3585 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }

```

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

```

3586 \bool_set_true:N \l_@@_in_code_after_bool
3587 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3588 \scan_stop:
3589 \tl_gclear:N \g_nicematrix_code_after_tl

```

```

3590 \group_end:

\g_@@_pre_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.
3591 \tl_if_empty:NF \g_@@_pre_code_before_tl
3592 {
3593   \tl_gput_right:Nx \g_@@_aux_tl
3594   {
3595     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3596     { \exp_not:V \g_@@_pre_code_before_tl }
3597   }
3598   \tl_gclear:N \g_@@_pre_code_before_tl
3599 }
3600 \tl_if_empty:NF \g_nicematrix_code_before_tl
3601 {
3602   \tl_gput_right:Nx \g_@@_aux_tl
3603   {
3604     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3605     { \exp_not:V \g_nicematrix_code_before_tl }
3606   }
3607   \tl_gclear:N \g_nicematrix_code_before_tl
3608 }

3609 \str_gclear:N \g_@@_name_env_str
3610 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>77</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 end of arrays done by `colortbl`.

```

3611 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3612 }

```

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`.

```

3613 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3614 { \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).

```

3615 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3616 {
3617   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3618   { \@@_adjust_pos_of_blocks_seq_i:nmnn #1 }
3619 }

```

The following command must *not* be protected.

```

3620 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nmnn #1 #2 #3 #4 #5
3621 {
3622   { #1 }
3623   { #2 }
3624   {
3625     \int_compare:nNnTF { #3 } > { 99 }

```

---

<sup>77</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```

3626     { \int_use:N \c@iRow }
3627     { #3 }
3628   }
3629   {
3630     \int_compare:nNnTF { #4 } > { 99 }
3631     { \int_use:N \c@jCol }
3632     { #4 }
3633   }
3634   { #5 }
3635 }

```

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).

```

3636 \hook_gput_code:nmn { begindocument } { . }
3637 {
3638   \cs_new_protected:Npx \@@_draw_dotted_lines:
3639   {
3640     \c_@@_pgfortikzpicture_tl
3641     \@@_draw_dotted_lines_i:
3642     \c_@@_endpgfortikzpicture_tl
3643   }
3644 }

```

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

```

3645 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3646 {
3647   \pgfrememberpicturepositiononpagetrue
3648   \pgf@relevantforpicturesizefalse
3649   \g_@@_HVdotsfor_lines_tl
3650   \g_@@_Vdots_lines_tl
3651   \g_@@_Ddots_lines_tl
3652   \g_@@_Iddots_lines_tl
3653   \g_@@_Cdots_lines_tl
3654   \g_@@_Ldots_lines_tl
3655 }

3656 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3657 {
3658   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3659   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3660 }

```

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

```

3661 \pgfdeclareshape { @@_diag_node }
3662 {
3663   \savedanchor { \five }
3664   {
3665     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3666     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3667   }
3668   \anchor { 5 } { \five }
3669   \anchor { center } { \pgfpointorigin }
3670 }

```

The following command creates the diagonal nodes (in fact, if the matrix is not a square matrix, not all the nodes are on the diagonal).

```

3671 \cs_new_protected:Npn \@@_create_diag_nodes:
3672 {
3673   \pgfpicture

```



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.

```
3712 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3713 {
```

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

```
3714 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3715 \int_set:Nn \l_@@_initial_i_int { #1 }
3716 \int_set:Nn \l_@@_initial_j_int { #2 }
3717 \int_set:Nn \l_@@_final_i_int { #1 }
3718 \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.

```
3719 \bool_set_false:N \l_@@_stop_loop_bool
3720 \bool_do_until:Nn \l_@@_stop_loop_bool
3721 {
3722   \int_add:Nn \l_@@_final_i_int { #3 }
3723   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3724   \bool_set_false:N \l_@@_final_open_bool
3725   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3726   {
3727     \int_compare:nNnTF { #3 } = 1
3728     { \bool_set_true:N \l_@@_final_open_bool }
3729     {
3730       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3731       { \bool_set_true:N \l_@@_final_open_bool }
3732     }
3733   }
3734   {
3735     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3736     {
3737       \int_compare:nNnT { #4 } = { -1 }
3738       { \bool_set_true:N \l_@@_final_open_bool }
3739     }
3740     {
3741       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3742       {
3743         \int_compare:nNnT { #4 } = 1
3744         { \bool_set_true:N \l_@@_final_open_bool }
3745       }
3746     }
3747   }
3748   \bool_if:NTF \l_@@_final_open_bool
```

If we are outside the matrix, we have found the extremity of the dotted line and it’s an *open* extremity.

```
3749   {
```

We do a step backwards.

```
3750     \int_sub:Nn \l_@@_final_i_int { #3 }
3751     \int_sub:Nn \l_@@_final_j_int { #4 }
3752     \bool_set_true:N \l_@@_stop_loop_bool
3753   }
```



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`.

```

3754     {
3755         \cs_if_exist:cTF
3756         {
3757             @@ _ dotted _
3758             \int_use:N \l_@@_final_i_int -
3759             \int_use:N \l_@@_final_j_int
3760         }
3761         {
3762             \int_sub:Nn \l_@@_final_i_int { #3 }
3763             \int_sub:Nn \l_@@_final_j_int { #4 }
3764             \bool_set_true:N \l_@@_final_open_bool
3765             \bool_set_true:N \l_@@_stop_loop_bool
3766         }
3767         {
3768             \cs_if_exist:cTF
3769             {
3770                 pgf @ sh @ ns @ \@@_env:
3771                 - \int_use:N \l_@@_final_i_int
3772                 - \int_use:N \l_@@_final_j_int
3773             }
3774             { \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.

```

3775         {
3776             \cs_set:cpn
3777             {
3778                 @@ _ dotted _
3779                 \int_use:N \l_@@_final_i_int -
3780                 \int_use:N \l_@@_final_j_int
3781             }
3782             { }
3783         }
3784     }
3785 }
3786 }

```

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

```

3787     \bool_set_false:N \l_@@_stop_loop_bool
3788     \bool_do_until:Nn \l_@@_stop_loop_bool
3789     {
3790         \int_sub:Nn \l_@@_initial_i_int { #3 }
3791         \int_sub:Nn \l_@@_initial_j_int { #4 }
3792         \bool_set_false:N \l_@@_initial_open_bool
3793         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3794         {
3795             \int_compare:nNnTF { #3 } = 1
3796             { \bool_set_true:N \l_@@_initial_open_bool }
3797             {
3798                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
3799                 { \bool_set_true:N \l_@@_initial_open_bool }
3800             }
3801         }
3802         {
3803             \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3804             {

```

```

3805         \int_compare:nNnT { #4 } = 1
3806         { \bool_set_true:N \l_@@_initial_open_bool }
3807     }
3808     {
3809         \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3810         {
3811             \int_compare:nNnT { #4 } = { -1 }
3812             { \bool_set_true:N \l_@@_initial_open_bool }
3813         }
3814     }
3815 }
3816 \bool_if:NTF \l_@@_initial_open_bool
3817 {
3818     \int_add:Nn \l_@@_initial_i_int { #3 }
3819     \int_add:Nn \l_@@_initial_j_int { #4 }
3820     \bool_set_true:N \l_@@_stop_loop_bool
3821 }
3822 {
3823     \cs_if_exist:cTF
3824     {
3825         @@ _ dotted _
3826         \int_use:N \l_@@_initial_i_int -
3827         \int_use:N \l_@@_initial_j_int
3828     }
3829     {
3830         \int_add:Nn \l_@@_initial_i_int { #3 }
3831         \int_add:Nn \l_@@_initial_j_int { #4 }
3832         \bool_set_true:N \l_@@_initial_open_bool
3833         \bool_set_true:N \l_@@_stop_loop_bool
3834     }
3835     {
3836         \cs_if_exist:cTF
3837         {
3838             pgf @ sh @ ns @ \@@_env:
3839             - \int_use:N \l_@@_initial_i_int
3840             - \int_use:N \l_@@_initial_j_int
3841         }
3842         { \bool_set_true:N \l_@@_stop_loop_bool }
3843         {
3844             \cs_set:cpn
3845             {
3846                 @@ _ dotted _
3847                 \int_use:N \l_@@_initial_i_int -
3848                 \int_use:N \l_@@_initial_j_int
3849             }
3850             { }
3851         }
3852     }
3853 }
3854 }

```

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.

```

3855     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3856     {
3857         { \int_use:N \l_@@_initial_i_int }

```

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

```

3858     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3859     { \int_use:N \l_@@_final_i_int }
3860     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3861     { } % for the name of the block
3862 }

```

```
3863 }
```

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).

```
3864 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3865 {
3866   \int_set:Nn \l_@@_row_min_int 1
3867   \int_set:Nn \l_@@_col_min_int 1
3868   \int_set_eq:NN \l_@@_row_max_int \c@iRow
3869   \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`.

```
3870   \seq_map_inline:Nn \g_@@_submatrix_seq
3871   { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
3872 }
```

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

```
3873 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
3874 {
3875   \bool_if:nT
3876   {
3877     \int_compare_p:n { #3 <= #1 }
3878     && \int_compare_p:n { #1 <= #5 }
3879     && \int_compare_p:n { #4 <= #2 }
3880     && \int_compare_p:n { #2 <= #6 }
3881   }
3882   {
3883     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3884     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3885     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3886     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3887   }
3888 }
```

```
3889 \cs_new_protected:Npn \@@_set_initial_coords:
3890 {
3891   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3892   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3893 }
3894 \cs_new_protected:Npn \@@_set_final_coords:
3895 {
3896   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3897   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3898 }
3899 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3900 {
3901   \pgfpointanchor
3902   {
3903     \@@_env:
3904     - \int_use:N \l_@@_initial_i_int
3905     - \int_use:N \l_@@_initial_j_int
3906   }
3907   { #1 }
3908   \@@_set_initial_coords:
3909 }
3910 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3911 {
3912   \pgfpointanchor
```

```

3913     {
3914       \@@_env:
3915       - \int_use:N \l_@@_final_i_int
3916       - \int_use:N \l_@@_final_j_int
3917     }
3918     { #1 }
3919     \@@_set_final_coords:
3920   }

3921 \cs_new_protected:Npn \@@_open_x_initial_dim:
3922 {
3923   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3924   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3925   {
3926     \cs_if_exist:cT
3927     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3928     {
3929       \pgfpointanchor
3930       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3931       { west }
3932       \dim_set:Nn \l_@@_x_initial_dim
3933       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3934     }
3935   }

```

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

```

3936   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3937   {
3938     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3939     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3940     \dim_add:Nn \l_@@_x_initial_dim \col@sep
3941   }
3942 }

```

```

3943 \cs_new_protected:Npn \@@_open_x_final_dim:
3944 {
3945   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3946   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3947   {
3948     \cs_if_exist:cT
3949     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3950     {
3951       \pgfpointanchor
3952       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3953       { east }
3954       \dim_set:Nn \l_@@_x_final_dim
3955       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3956     }
3957   }

```

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

```

3958   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3959   {
3960     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
3961     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3962     \dim_sub:Nn \l_@@_x_final_dim \col@sep
3963   }
3964 }

```

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.

```

3965 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3966 {
3967   \@@_adjust_to_submatrix:nn { #1 } { #2 }

```

```

3968 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3969 {
3970 \@@_find_extremities_of_line:nmmn { #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.

```

3971 \group_begin:
3972 \int_compare:nNnTF { #1 } = 0
3973 { \color { nicematrix-first-row } }
3974 {

```

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.

```

3975 \int_compare:nNnT { #1 } = \l_@@_last_row_int
3976 { \color { nicematrix-last-row } }
3977 }
3978 \keys_set:nn { NiceMatrix / xdots } { #3 }
3979 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3980 \@@_actually_draw_Ldots:
3981 \group_end:
3982 }
3983 }

```

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`.

```

3984 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3985 {
3986 \bool_if:NTF \l_@@_initial_open_bool
3987 {
3988 \@@_open_x_initial_dim:
3989 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3990 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3991 }
3992 { \@@_set_initial_coords_from_anchor:n { base~east } }
3993 \bool_if:NTF \l_@@_final_open_bool
3994 {
3995 \@@_open_x_final_dim:
3996 \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3997 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3998 }
3999 { \@@_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 (?).

```

4000 \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4001 \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4002 \@@_draw_line:
4003 }

```

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.

```

4004 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4005 {
4006   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4007   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4008   {
4009     \@@_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.

```

4010   \group_begin:
4011     \int_compare:nNnTF { #1 } = 0
4012     { \color { nicematrix-first-row } }
4013     {

```

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.

```

4014       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4015       { \color { nicematrix-last-row } }
4016     }
4017     \keys_set:nn { NiceMatrix / xdots } { #3 }
4018     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4019     \@@_actually_draw_Cdots:
4020   \group_end:
4021 }
4022 }

```

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`.

```

4023 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4024 {
4025   \bool_if:NTF \l_@@_initial_open_bool
4026   { \@@_open_x_initial_dim: }
4027   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4028   \bool_if:NTF \l_@@_final_open_bool
4029   { \@@_open_x_final_dim: }
4030   { \@@_set_final_coords_from_anchor:n { mid-west } }
4031   \bool_lazy_and:nnTF
4032     \l_@@_initial_open_bool
4033     \l_@@_final_open_bool
4034   {
4035     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4036     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4037     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4038     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4039     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4040   }
4041   {
4042     \bool_if:NT \l_@@_initial_open_bool
4043     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4044     \bool_if:NT \l_@@_final_open_bool
4045     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4046   }
4047   \@@_draw_line:
4048 }

```

```

4049 \cs_new_protected:Npn \@@_open_y_initial_dim:
4050 {
4051   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4052   \dim_set:Nn \l_@@_y_initial_dim
4053   {
4054     \fp_to_dim:n
4055     {
4056       \pgf@y
4057       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4058     }
4059   } % modified 6.13c
4060   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4061   {
4062     \cs_if_exist:cT
4063     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4064     {
4065       \pgfpointanchor
4066       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4067       { north }
4068       \dim_set:Nn \l_@@_y_initial_dim
4069       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4070     }
4071   }
4072 }
4073 \cs_new_protected:Npn \@@_open_y_final_dim:
4074 {
4075   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4076   \dim_set:Nn \l_@@_y_final_dim
4077   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4078   % modified 6.13c
4079   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4080   {
4081     \cs_if_exist:cT
4082     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4083     {
4084       \pgfpointanchor
4085       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4086       { south }
4087       \dim_set:Nn \l_@@_y_final_dim
4088       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4089     }
4090   }
4091 }

```

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.

```

4092 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4093 {
4094   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4095   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4096   {
4097     \@@_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.

```

4098   \group_begin:
4099   \int_compare:nNnTF { #2 } = 0
4100   { \color { nicematrix-first-col } }
4101   {
4102     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4103     { \color { nicematrix-last-col } }
4104   }
4105   \keys_set:nn { NiceMatrix / xdots } { #3 }
4106   \tl_if_empty:VF \l_@@_xdots_color_tl

```

```

4107         { \color { \l_@@_xdots_color_tl } }
4108     \@@_actually_draw_Vdots:
4109     \group_end:
4110 }
4111 }

```

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`.

```

4112 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4113 {

```

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

```

4114     \bool_set_false:N \l_tmpa_bool

```

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

```

4115     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
4116     {
4117         \@@_set_initial_coords_from_anchor:n { south-west }
4118         \@@_set_final_coords_from_anchor:n { north-west }
4119         \bool_set:Nn \l_tmpa_bool
4120         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4121     }

```

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

```

4122     \bool_if:NTF \l_@@_initial_open_bool
4123     \@@_open_y_initial_dim:
4124     { \@@_set_initial_coords_from_anchor:n { south } }
4125     \bool_if:NTF \l_@@_final_open_bool
4126     \@@_open_y_final_dim:
4127     { \@@_set_final_coords_from_anchor:n { north } }
4128     \bool_if:NTF \l_@@_initial_open_bool
4129     {
4130         \bool_if:NTF \l_@@_final_open_bool
4131         {
4132             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4133             \dim_set_eq:NN \l_tmpa_dim \pgf@x
4134             \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4135             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4136             \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.

```

4137     \int_compare:nNnT \l_@@_last_col_int > { -2 }
4138     {
4139         \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
4140         {
4141             \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
4142             \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
4143             \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4144             \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
4145         }
4146     }

```



```

4147     }
4148     { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
4149   }
4150   {
4151     \bool_if:NTF \l_@@_final_open_bool
4152     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
4153     {

```

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.

```

4154         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4155         {
4156           \dim_set:Nn \l_@@_x_initial_dim
4157           {
4158             \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4159             \l_@@_x_initial_dim \l_@@_x_final_dim
4160           }
4161           \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4162         }
4163       }
4164     }
4165   \@@_draw_line:
4166 }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonal 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.

```

4167 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4168 {
4169   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4170   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4171   {
4172     \@@_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.

```

4173   \group_begin:
4174     \keys_set:nn { NiceMatrix / xdots } { #3 }
4175     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4176     \@@_actually_draw_Ddots:
4177   \group_end:
4178 }
4179 }

```

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`.

```

4180 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4181 {
4182   \bool_if:NTF \l_@@_initial_open_bool
4183   {
4184     \@@_open_y_initial_dim:
4185     \@@_open_x_initial_dim:
4186   }
4187   { \@@_set_initial_coords_from_anchor:n { south-east } }
4188   \bool_if:NTF \l_@@_final_open_bool
4189   {
4190     \@@_open_x_final_dim:
4191     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4192   }
4193   { \@@_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.

```

4194   \bool_if:NT \l_@@_parallelize_diags_bool
4195   {
4196     \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).

```

4197     \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.

```

4198     {
4199       \dim_gset:Nn \g_@@_delta_x_one_dim
4200       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4201       \dim_gset:Nn \g_@@_delta_y_one_dim
4202       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4203     }

```

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`.

```

4204     {
4205       \dim_set:Nn \l_@@_y_final_dim
4206       {
4207         \l_@@_y_initial_dim +
4208         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4209         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4210       }
4211     }
4212   }
4213   \@@_draw_line:
4214 }

```

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.

```

4215 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4216 {
4217   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4218   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4219   {
4220     \@@_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.

```

4221     \group_begin:
4222     \keys_set:nn { NiceMatrix / xdots } { #3 }
4223     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4224     \@@_actually_draw_Iddots:

```

```

4225     \group_end:
4226   }
4227 }

```

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.`

```

4228 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4229 {
4230   \bool_if:NTF \l_@@_initial_open_bool
4231   {
4232     \@@_open_y_initial_dim:
4233     \@@_open_x_initial_dim:
4234   }
4235   { \@@_set_initial_coords_from_anchor:n { south~west } }
4236   \bool_if:NTF \l_@@_final_open_bool
4237   {
4238     \@@_open_y_final_dim:
4239     \@@_open_x_final_dim:
4240   }
4241   { \@@_set_final_coords_from_anchor:n { north~east } }
4242   \bool_if:NT \l_@@_parallelize_diags_bool
4243   {
4244     \int_gincr:N \g_@@_iddots_int
4245     \int_compare:nNnTF \g_@@_iddots_int = 1
4246     {
4247       \dim_gset:Nn \g_@@_delta_x_two_dim
4248       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4249       \dim_gset:Nn \g_@@_delta_y_two_dim
4250       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4251     }
4252     {
4253       \dim_set:Nn \l_@@_y_final_dim
4254       {
4255         \l_@@_y_initial_dim +
4256         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4257         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4258       }
4259     }
4260   }
4261   \@@_draw_line:
4262 }

```

## The actual instructions for drawing the dotted lines with Tikz

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

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`

- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

4263 \cs_new_protected:Npn \@@_draw_line:
4264 {
4265   \pgfrememberpicturepositiononpagetrue
4266   \pgf@relevantforpicturesizefalse
4267   \bool_lazy_or:nnTF
4268   { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4269   \l_@@_dotted_bool
4270   \@@_draw_standard_dotted_line:
4271   \@@_draw_unstandard_dotted_line:
4272 }

```

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.

```

4273 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4274 {
4275   \begin { scope }
4276   \@@_draw_unstandard_dotted_line:o
4277   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4278 }

```

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

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

```

4279 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4280 {
4281   \@@_draw_unstandard_dotted_line:nVV
4282   { #1 }
4283   \l_@@_xdots_up_tl
4284   \l_@@_xdots_down_tl
4285 }
4286 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4287 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
4288 {
4289   \draw
4290   [
4291     #1 ,
4292     shorten-> = \l_@@_xdots_shorten_end_dim ,
4293     shorten-< = \l_@@_xdots_shorten_start_dim ,
4294   ]
4295   ( \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).

```

4296   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
4297   node [ sloped , below ] { $ \scriptstyle #3 $ }
4298   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4299 \end { scope }
4300 }
4301 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnn { n V V }

```

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

```

4302 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4303 {
4304   \bool_lazy_and:nnF
4305   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4306   { \tl_if_empty_p:N \l_@@_xdots_down_tl }

```

```

4307 {
4308   \pgfscope
4309   \pgftransformshift
4310   {
4311     \pgfpointlineattime { 0.5 }
4312     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4313     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4314   }
4315   \pgftransformrotate
4316   {
4317     \fp_eval:n
4318     {
4319       atand
4320       (
4321         \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4322         \l_@@_x_final_dim - \l_@@_x_initial_dim
4323       )
4324     }
4325   }
4326   \pgfnode
4327   { rectangle }
4328   { south }
4329   {
4330     \c_math_toggle_token
4331     \scriptstyle \l_@@_xdots_up_tl
4332     \c_math_toggle_token
4333   }
4334   { }
4335   { \pgfusepath { } }
4336   \pgfnode
4337   { rectangle }
4338   { north }
4339   {
4340     \c_math_toggle_token
4341     \scriptstyle \l_@@_xdots_down_tl
4342     \c_math_toggle_token
4343   }
4344   { }
4345   { \pgfusepath { } }
4346   \endpgfscope
4347 }
4348 \group_begin:

```

The dimension  $\l_@@_l\_dim$  is the length  $\ell$  of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4349   \dim_zero_new:N \l_@@_l_dim
4350   \dim_set:Nn \l_@@_l_dim
4351   {
4352     \fp_to_dim:n
4353     {
4354       sqrt
4355       (
4356         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4357         +
4358         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4359       )
4360     }
4361   }

```

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.

```

4362   \bool_lazy_or:nnF

```

```

4363     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4364     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4365     \@@_draw_standard_dotted_line_i:
4366   \group_end:
4367 }
4368 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4369 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4370 {

```

The number of dots will be  $\l_1\text{tmpa\_int} + 1$ .

```

4371   \bool_if:NTF \l_@@_initial_open_bool
4372   {
4373     \bool_if:NTF \l_@@_final_open_bool
4374     {
4375       \int_set:Nn \l_tmpa_int
4376       { \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim }
4377     }
4378     {
4379       \int_set:Nn \l_tmpa_int
4380       {
4381         \dim_ratio:nn
4382         { \l_@@_l_dim - \l_@@_xdots_shorten_start_dim }
4383         \l_@@_xdots_inter_dim
4384       }
4385     }
4386   }
4387   {
4388     \bool_if:NTF \l_@@_final_open_bool
4389     {
4390       \int_set:Nn \l_tmpa_int
4391       {
4392         \dim_ratio:nn
4393         { \l_@@_l_dim - \l_@@_xdots_shorten_end_dim }
4394         \l_@@_xdots_inter_dim
4395       }
4396     }
4397     {
4398       \int_set:Nn \l_tmpa_int
4399       {
4400         \dim_ratio:nn
4401         {
4402           \l_@@_l_dim
4403           - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4404         }
4405         \l_@@_xdots_inter_dim
4406       }
4407     }
4408   }

```

The dimensions  $\l_1\text{tmpa\_dim}$  and  $\l_1\text{tmpb\_dim}$  are the coordinates of the vector between two dots in the dotted line.

```

4409   \dim_set:Nn \l_tmpa_dim
4410   {
4411     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4412     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4413   }
4414   \dim_set:Nn \l_tmpb_dim
4415   {
4416     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4417     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4418   }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4419 \dim_gadd:Nn \l_@@_x_initial_dim
4420 {
4421   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4422   \dim_ratio:nn
4423   {
4424     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4425     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4426   }
4427   { 2 \l_@@_l_dim }
4428 }
4429 \dim_gadd:Nn \l_@@_y_initial_dim
4430 {
4431   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4432   \dim_ratio:nn
4433   {
4434     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4435     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4436   }
4437   { 2 \l_@@_l_dim }
4438 }
4439 \pgf@relevantforpicturesizefalse
4440 \int_step_inline:nnn 0 \l_tmpa_int
4441 {
4442   \pgfpathcircle
4443   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4444   { \l_@@_xdots_radius_dim }
4445   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4446   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4447 }
4448 \pgfusepathqfill
4449 }

```

## User commands available in the new environments

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

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

4450 \hook_gput_code:nnn { begindocument } { . }
4451 {
4452   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
4453   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4454   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
4455   {
4456     \int_compare:nNnTF \c@jCol = 0
4457     { \@@_error:nn { in-first~col } \Ldots }
4458     {
4459       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4460       { \@@_error:nn { in-last~col } \Ldots }
4461       {
4462         \@@_instruction_of_type:nnn \c_false_bool { Ldots }

```

```

4463         { #1 , down = #2 , up = #3 }
4464     }
4465 }
4466 \bool_if:NF \l_@@_nullify_dots_bool
4467 { \phantom { \ensuremath { \@@_old_ldots } } }
4468 \bool_gset_true:N \g_@@_empty_cell_bool
4469 }

4470 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4471 {
4472     \int_compare:nNnTF \c@jCol = 0
4473     { \@@_error:nn { in~first~col } \Cdots }
4474     {
4475         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4476         { \@@_error:nn { in~last~col } \Cdots }
4477         {
4478             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4479             { #1 , down = #2 , up = #3 }
4480         }
4481     }
4482     \bool_if:NF \l_@@_nullify_dots_bool
4483     { \phantom { \ensuremath { \@@_old_cdots } } }
4484     \bool_gset_true:N \g_@@_empty_cell_bool
4485 }

4486 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4487 {
4488     \int_compare:nNnTF \c@iRow = 0
4489     { \@@_error:nn { in~first~row } \Vdots }
4490     {
4491         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4492         { \@@_error:nn { in~last~row } \Vdots }
4493         {
4494             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4495             { #1 , down = #2 , up = #3 }
4496         }
4497     }
4498     \bool_if:NF \l_@@_nullify_dots_bool
4499     { \phantom { \ensuremath { \@@_old_vdots } } }
4500     \bool_gset_true:N \g_@@_empty_cell_bool
4501 }

4502 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4503 {
4504     \int_case:nnF \c@iRow
4505     {
4506         0 { \@@_error:nn { in~first~row } \Ddots }
4507         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4508     }
4509     {
4510         \int_case:nnF \c@jCol
4511         {
4512             0 { \@@_error:nn { in~first~col } \Ddots }
4513             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4514         }
4515         {
4516             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4517             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4518             { #1 , down = #2 , up = #3 }
4519         }
4520     }

```



```

4521     }
4522     \bool_if:NF \l_@@_nullify_dots_bool
4523     { \phantom { \ensuremath { \@@_old_ddots } } }
4524     \bool_gset_true:N \g_@@_empty_cell_bool
4525     }

4526 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4527 {
4528     \int_case:nnF \c@iRow
4529     {
4530         0 { \@@_error:nn { in~first~row } \Iddots }
4531         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4532     }
4533     {
4534         \int_case:nnF \c@jCol
4535         {
4536             0 { \@@_error:nn { in~first~col } \Iddots }
4537             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4538         }
4539         {
4540             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4541             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4542             { #1 , down = #2 , up = #3 }
4543         }
4544     }
4545     \bool_if:NF \l_@@_nullify_dots_bool
4546     { \phantom { \ensuremath { \@@_old_iddots } } }
4547     \bool_gset_true:N \g_@@_empty_cell_bool
4548     }
4549 }

```

End of the `\AddToHook`.

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

```

4550 \keys_define:nn { NiceMatrix / Ddots }
4551 {
4552     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4553     draw-first .default:n = true ,
4554     draw-first .value_forbidden:n = true
4555 }

```

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

```

4556 \cs_new_protected:Npn \@@_Hspace:
4557 {
4558     \bool_gset_true:N \g_@@_empty_cell_bool
4559     \hspace
4560 }

```

In the environments of `nicematrix`, the command `\multicolumn` is redefined. We will patch the environment `{tabular}` to go back to the previous value of `\multicolumn`.

```

4561 \cs_set_eq:NN \@@_old_multicolumn \multicolumn

```

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

This command must *not* be protected since it begins with `\multicolumn`.

```

4562 \cs_new:Npn \@@_Hdotsfor:
4563 {
4564     \bool_lazy_and:nnTF
4565     { \int_compare_p:nNn \c@jCol = 0 }
4566     { \int_compare_p:nNn \l_@@_first_col_int = 0 }

```

```

4567 {
4568   \bool_if:NTF \g_@@_after_col_zero_bool
4569   {
4570     \multicolumn { 1 } { c } { }
4571     \@@_Hdotsfor_i
4572   }
4573   { \@@_fatal:n { Hdotsfor~in~col~0 } }
4574 }
4575 {
4576   \multicolumn { 1 } { c } { }
4577   \@@_Hdotsfor_i
4578 }
4579 }

```

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:`).

```

4580 \hook_gput_code:nnn { begindocument } { . }
4581 {
4582   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4583   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put `!` before the last optional argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

4584   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4585   {
4586     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4587     {
4588       \@@_Hdotsfor:nnnn
4589       { \int_use:N \c@iRow }
4590       { \int_use:N \c@jCol }
4591       { #2 }
4592       {
4593         #1 , #3 ,
4594         down = \exp_not:n { #4 } ,
4595         up = \exp_not:n { #5 }
4596       }
4597     }
4598     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4599   }
4600 }

```

```

4601 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4602 {
4603   \bool_set_false:N \l_@@_initial_open_bool
4604   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4605   \int_set:Nn \l_@@_initial_i_int { #1 }
4606   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4607   \int_compare:nNnTF { #2 } = 1
4608   {
4609     \int_set:Nn \l_@@_initial_j_int 1
4610     \bool_set_true:N \l_@@_initial_open_bool
4611   }
4612   {
4613     \cs_if_exist:cTF
4614     {
4615       pgf @ sh @ ns @ \@@_env:
4616       - \int_use:N \l_@@_initial_i_int
4617       - \int_eval:n { #2 - 1 }
4618     }
4619     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }

```

```

4620     {
4621         \int_set:Nn \l_@@_initial_j_int { #2 }
4622         \bool_set_true:N \l_@@_initial_open_bool
4623     }
4624 }
4625 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4626 {
4627     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4628     \bool_set_true:N \l_@@_final_open_bool
4629 }
4630 {
4631     \cs_if_exist:cTF
4632     {
4633         pgf @ sh @ ns @ \@@_env:
4634         - \int_use:N \l_@@_final_i_int
4635         - \int_eval:n { #2 + #3 }
4636     }
4637     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4638     {
4639         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4640         \bool_set_true:N \l_@@_final_open_bool
4641     }
4642 }
4643 \group_begin:
4644 \int_compare:nNnTF { #1 } = 0
4645 { \color { nicematrix-first-row } }
4646 {
4647     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4648     { \color { nicematrix-last-row } }
4649 }
4650 \keys_set:mn { NiceMatrix / xdots } { #4 }
4651 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4652 \@@_actually_draw_Ldots:
4653 \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).

```

4654     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4655     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4656 }

4657 \hook_gput_code:nnn { begindocument } { . }
4658 {
4659     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4660     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4661     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4662     {
4663         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4664         {
4665             \@@_Vdotsfor:nnnn
4666             { \int_use:N \c@iRow }
4667             { \int_use:N \c@jCol }
4668             { #2 }
4669             {
4670                 #1 , #3 ,
4671                 down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4672             }
4673         }
4674     }
4675 }

```

Enf of \AddToHook.

```

4676 \cs_new_protected:Npn \l_@@_Vdotsfor:nmmn #1 #2 #3 #4
4677   {
4678     \bool_set_false:N \l_@@_initial_open_bool
4679     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

4680     \int_set:Nn \l_@@_initial_j_int { #2 }
4681     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

4682     \int_compare:nNnTF #1 = 1
4683     {
4684       \int_set:Nn \l_@@_initial_i_int 1
4685       \bool_set_true:N \l_@@_initial_open_bool
4686     }
4687     {
4688       \cs_if_exist:cTF
4689       {
4690         pgf @ sh @ ns @ \l_@@_env:
4691         - \int_eval:n { #1 - 1 }
4692         - \int_use:N \l_@@_initial_j_int
4693       }
4694       { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4695       {
4696         \int_set:Nn \l_@@_initial_i_int { #1 }
4697         \bool_set_true:N \l_@@_initial_open_bool
4698       }
4699     }
4700     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4701     {
4702       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4703       \bool_set_true:N \l_@@_final_open_bool
4704     }
4705     {
4706       \cs_if_exist:cTF
4707       {
4708         pgf @ sh @ ns @ \l_@@_env:
4709         - \int_eval:n { #1 + #3 }
4710         - \int_use:N \l_@@_final_j_int
4711       }
4712       { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4713       {
4714         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4715         \bool_set_true:N \l_@@_final_open_bool
4716       }
4717     }
4718     \group_begin:
4719     \int_compare:nNnTF { #2 } = 0
4720     { \color { nicematrix-first-col } }
4721     {
4722       \int_compare:nNnTF { #2 } = \g_@@_col_total_int
4723       { \color { nicematrix-last-col } }
4724     }
4725     \keys_set:nn { NiceMatrix / xdots } { #4 }
4726     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4727     \l_@@_actually_draw_Vdots:
4728     \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 \l\_@@\_find\_extremities\_of\_line:nmmn). This declaration is done by defining a special control sequence (to nil).

```

4729     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }

```

```

4730     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4731 }

```

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

```

4732 \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 the following behaviour:

- If the argument is of the format  $i$ - $j$ , our command applies the command `\int_eval:n` to  $i$  and  $j$  ;
- If not (that is to say, when it's a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).<sup>78</sup>

```

4733 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4734 {
4735   \tl_if_empty:nTF { #2 }
4736     { #1 }
4737     { \@@_double_int_eval_i:n #1-#2 \q_stop }
4738 }
4739 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
4740 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

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

```

4741 \hook_gput_code:nnn { begindocument } { . }
4742 {
4743   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4744   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4745   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4746     {
4747       \group_begin:
4748       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4749       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4750       \use:e
4751       {
4752         \@@_line_i:nn
4753         { \@@_double_int_eval:n #2 - \q_stop }
4754         { \@@_double_int_eval:n #3 - \q_stop }
4755       }
4756       \group_end:
4757     }
4758 }
4759 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4760 {
4761   \bool_set_false:N \l_@@_initial_open_bool
4762   \bool_set_false:N \l_@@_final_open_bool
4763   \bool_if:nTF

```

---

<sup>78</sup>Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

4764     {
4765     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4766     ||
4767     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4768     }
4769     {
4770     \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4771     }
4772     { \@@_draw_line_ii:nn { #1 } { #2 } }
4773   }
4774 \hook_gput_code:nnn { begindocument } { . }
4775   {
4776     \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4777     {

```

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:`.

```

4778     \c_@@_pgfortikzpicture_tl
4779     \@@_draw_line_iii:nn { #1 } { #2 }
4780     \c_@@_endpgfortikzpicture_tl
4781   }
4782 }

```

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

```

4783 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4784   {
4785     \pgfrememberpicturepositiononpagetrue
4786     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4787     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4788     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4789     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4790     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4791     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4792     \@@_draw_line:
4793   }

```

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

## The command `\RowStyle`

```

4794 \keys_define:nn { NiceMatrix / RowStyle }
4795   {
4796     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4797     cell-space-top-limit .initial:n = \c_zero_dim ,
4798     cell-space-top-limit .value_required:n = true ,
4799     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4800     cell-space-bottom-limit .initial:n = \c_zero_dim ,
4801     cell-space-bottom-limit .value_required:n = true ,
4802     cell-space-limits .meta:n =
4803     {
4804       cell-space-top-limit = #1 ,
4805       cell-space-bottom-limit = #1 ,
4806     } ,
4807     color .tl_set:N = \l_@@_color_tl ,
4808     color .value_required:n = true ,
4809     bold .bool_set:N = \l_tmpa_bool ,
4810     bold .default:n = true ,
4811     bold .initial:n = false ,
4812     nb-rows .code:n =

```

```

4813     \str_if_eq:nnTF { #1 } { * }
4814     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
4815     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
4816     nb-rows .value_required:n = true ,
4817     rowcolor .tl_set:N = \l_tmpa_tl ,
4818     rowcolor .value_required:n = true ,
4819     rowcolor .initial:n = ,
4820     unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
4821 }

```

```

4822 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4823 {
4824   \group_begin:
4825   \tl_clear:N \l_tmpa_tl % value of \rowcolor
4826   \tl_clear:N \l_@@_color_tl
4827   \int_set:Nn \l_@@_key_nb_rows_int 1
4828   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

4829   \tl_if_empty:NF \l_tmpa_tl
4830   {

```

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

```

4831     \tl_gput_right:Nx \g_@@_pre_code_before_tl
4832     {

```

The command `\@@_exp_color_arg:N` is *fully expandable*.

```

4833     \@@_exp_color_arg:N \@@_rectanglecolor \l_tmpa_tl
4834     { \int_use:N \c@iRow - \int_use:N \c@jCol }
4835     { \int_use:N \c@iRow - * }
4836   }

```

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

```

4837     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4838     {
4839       \tl_gput_right:Nx \g_@@_pre_code_before_tl
4840       {
4841         \@@_exp_color_arg:N \@@_rowcolor \l_tmpa_tl
4842         {
4843           \int_eval:n { \c@iRow + 1 }
4844           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4845         }
4846       }
4847     }
4848   }
4849   \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4850   \tl_gput_right:Nx \g_@@_row_style_tl
4851   { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4852   \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

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

```

4853   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4854   {
4855     \tl_gput_right:Nx \g_@@_row_style_tl
4856     {
4857       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4858       {
4859         \dim_set:Nn \l_@@_cell_space_top_limit_dim
4860         { \dim_use:N \l_tmpa_dim }
4861       }
4862     }
4863   }

```

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

```

4864   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4865   {
4866     \tl_gput_right:Nx \g_@@_row_style_tl

```

```

4867     {
4868         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
4869         {
4870             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4871             { \dim_use:N \l_tmpb_dim }
4872         }
4873     }
4874 }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

4875     \tl_if_empty:NF \l_@@_color_tl
4876     {
4877         \tl_gput_right:Nx \g_@@_row_style_tl
4878         {
4879             \mode_leave_vertical:
4880             \@@_color:n { \l_@@_color_tl }
4881         }
4882     }

```

`\l_tmpa_bool` is the value of the key `bold`.

```

4883     \bool_if:NT \l_tmpa_bool
4884     {
4885         \tl_gput_right:Nn \g_@@_row_style_tl
4886         {
4887             \if_mode_math:
4888                 \c_math_toggle_token
4889                 \bfseries \boldmath
4890                 \c_math_toggle_token
4891             \else:
4892                 \bfseries \boldmath
4893             \fi:
4894         }
4895     }
4896     \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4897     \group_end:
4898     \g_@@_row_style_tl
4899     \ignorespaces
4900 }

```

## Colors of cells, rows and columns

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

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

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

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

```

4901 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4902 {

```



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.

```
4903 \int_zero:N \l_tmpa_int
```

We don't take into account the colors like `myserie!+` because those colors are special color from a `\definecolorseries` of `xcolor`.

```
4904 \str_if_in:nnF { #1 } { !! }
4905 {
4906   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4907   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4908 }
4909 \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

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

```
4910 {
4911   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4912   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4913 }
```

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

```
4914 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4915 }
4916 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4917 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x x }
```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```
4918 \cs_new_protected:Npn \@@_actually_color:
4919 {
4920   \pgfpicture
4921   \pgf@relevantforpicturesizefalse
4922   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4923   {
4924     \color ##2
4925     \use:c { g_@@_color _ ##1 _tl }
4926     \tl_gclear:c { g_@@_color _ ##1 _tl }
4927     \pgfusepath { fill }
4928   }
4929   \endpgfpicture
4930 }
4931 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4932 {
4933   \tl_set:Nn \l_@@_rows_tl { #1 }
4934   \tl_set:Nn \l_@@_cols_tl { #2 }
4935   \@@_cartesian_path:
4936 }
```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
4937 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4938 {
4939   \tl_if_blank:nF { #2 }
4940   {
4941     \@@_add_to_colors_seq:xn
4942     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4943     { \@@_cartesian_color:nn { #3 } { - } }
4944   }
4945 }
```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

4946 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4947 {
4948   \tl_if_blank:nF { #2 }
4949   {
4950     \@@_add_to_colors_seq:xn
4951     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4952     { \@@_cartesian_color:nn { - } { #3 } }
4953   }
4954 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

4955 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4956 {
4957   \tl_if_blank:nF { #2 }
4958   {
4959     \@@_add_to_colors_seq:xn
4960     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4961     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4962   }
4963 }

```

The last argument is the radius of the corners of the rectangle.

```

4964 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4965 {
4966   \tl_if_blank:nF { #2 }
4967   {
4968     \@@_add_to_colors_seq:xn
4969     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4970     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4971   }
4972 }

```

The last argument is the radius of the corners of the rectangle.

```

4973 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4974 {
4975   \@@_cut_on_hyphen:w #1 \q_stop
4976   \tl_clear_new:N \l_@@_tmpc_tl
4977   \tl_clear_new:N \l_@@_tmpd_tl
4978   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
4979   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
4980   \@@_cut_on_hyphen:w #2 \q_stop
4981   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
4982   \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`.

```

4983   \@@_cartesian_path:n { #3 }
4984 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

4985 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4986 {
4987   \clist_map_inline:nn { #3 }
4988   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4989 }

```

```

4990 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4991 {
4992   \int_step_inline:nn { \int_use:N \c@iRow }

```

```

4993     {
4994         \int_step_inline:nn { \int_use:N \c@jCol }
4995         {
4996             \int_if_even:nTF { #####1 + #1 }
4997                 { \@@_cellcolor [ #1 ] { #2 } }
4998                 { \@@_cellcolor [ #1 ] { #3 } }
4999             { #1 - #####1 }
5000         }
5001     }
5002 }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns) and the cells in the “corners”.

```

5003 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5004 {
5005     \@@_rectanglecolor [ #1 ] { #2 }
5006     { 1 - 1 }
5007     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5008 }

```

```

5009 \keys_define:nn { NiceMatrix / rowcolors }
5010 {
5011     respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5012     respect-blocks .default:n = true ,
5013     cols .tl_set:N = \l_@@_cols_tl ,
5014     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5015     restart .default:n = true ,
5016     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5017 }

```

The command `\rowcolors` (accessible in the `code-before`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`. Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

**#1** (optional) is the color space ; **#2** is a list of intervals of rows ; **#3** is the list of colors ; **#4** is for the optional list of pairs *key=value*.

```

5018 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5019 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5020     \group_begin:
5021     \seq_clear_new:N \l_@@_colors_seq
5022     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5023     \tl_clear_new:N \l_@@_cols_tl
5024     \tl_set:Nn \l_@@_cols_tl { - }
5025     \keys_set:nn { NiceMatrix / rowcolors } { #4 }

```

The counter `\l_@@_color_int` will be the rank of the current color in the list of colors (modulo the length of the list).

```

5026     \int_zero_new:N \l_@@_color_int
5027     \int_set:Nn \l_@@_color_int 1
5028     \bool_if:NT \l_@@_respect_blocks_bool
5029     {

```

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`).

```

5030     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5031     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5032     { \@@_not_in_exterior_p:nnnn #1 }
5033 }

```

```

5034 \pgfpicture
5035 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5036 \clist_map_inline:nn { #2 }
5037 {
5038   \tl_set:Nn \l_tmpa_tl { ##1 }
5039   \tl_if_in:NnTF \l_tmpa_tl { - }
5040   { \@@_cut_on_hyphen:w ##1 \q_stop }
5041   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, `\l_tmpa_tl` and `\l_tmpb_tl` are the first row and the last row of the interval of rows that we have to treat. The counter `\l_tmpa_int` will be the index of the loop over the rows.

```

5042   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5043   \bool_if:NTF \l_@@_rowcolors_restart_bool
5044   { \int_set:Nn \l_@@_color_int 1 }
5045   { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
5046   \int_zero_new:N \l_@@_tmpc_int
5047   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5048   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5049   {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5050     \int_set_eq:NN \l_tmpb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

5051     \bool_if:NT \l_@@_respect_blocks_bool
5052     {
5053       \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5054       { \@@_intersect_our_row_p:nnnn ###1 }
5055       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5056     }
5057     \tl_set:Nx \l_@@_rows_tl
5058     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5059     \tl_clear_new:N \l_@@_color_tl
5060     \tl_set:Nx \l_@@_color_tl
5061     {
5062       \@@_color_index:n
5063       {
5064         \int_mod:nn
5065         { \l_@@_color_int - 1 }
5066         { \seq_count:N \l_@@_colors_seq }
5067         + 1
5068       }
5069     }
5070     \tl_if_empty:NF \l_@@_color_tl
5071     {
5072       \@@_add_to_colors_seq:xx
5073       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5074       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5075     }
5076     \int_incr:N \l_@@_color_int
5077     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5078   }
5079 }
5080 \endpgfpicture
5081 \group_end:
5082 }

```

The command `\@@_color_index:n` peeks in `\l_@@_colors_seq` the color at the index #1. However, if that color is the symbol `=`, the previous one is poken. This macro is recursive.

```

5083 \cs_new:Npn \@@_color_index:n #1

```

```

5084 {
5085   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5086   { \@@_color_index:n { #1 - 1 } }
5087   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5088 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

5089 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
5090 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

```

```

5091 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5092 {
5093   \int_compare:nNnT { #3 } > \l_tmpb_int
5094   { \int_set:Nn \l_tmpb_int { #3 } }
5095 }

```

```

5096 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5097 {
5098   \bool_lazy_or:nnTF
5099   { \int_compare_p:nNn { #4 } = \c_zero_int }
5100   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } } }
5101 \prg_return_false:
5102 \prg_return_true:
5103 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5104 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5105 {
5106   \bool_if:nTF
5107   {
5108     \int_compare_p:n { #1 <= \l_tmpa_int }
5109     &&
5110     \int_compare_p:n { \l_tmpa_int <= #3 }
5111   }
5112   \prg_return_true:
5113   \prg_return_false:
5114 }

```

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`).

```

5115 \cs_new_protected:Npn \@@_cartesian_path:n #1
5116 {
5117   \bool_lazy_and:nnT
5118   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5119   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5120   {
5121     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5122     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5123   }

```

We begin the loop over the columns.

```

5124 \clist_map_inline:Nn \l_@@_cols_tl
5125 {
5126   \tl_set:Nn \l_tmpa_tl { ##1 }
5127   \tl_if_in:NnTF \l_tmpa_tl { - }

```

```

5128     { \@@_cut_on_hyphen:w ##1 \q_stop }
5129     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5130 \bool_lazy_or:nnT
5131     { \tl_if_blank_p:V \l_tmpa_tl }
5132     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5133     { \tl_set:Nn \l_tmpa_tl { 1 } } }
5134 \bool_lazy_or:nnT
5135     { \tl_if_blank_p:V \l_tmpb_tl }
5136     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5137     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5138 \int_compare:nNnT \l_tmpb_tl > \c@jCol
5139     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5140     \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.

```

5141     \@@_qpoint:n { col - \l_tmpa_tl }
5142     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5143     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5144     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5145     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5146     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5147     \clist_map_inline:Nn \l_@@_rows_tl
5148     {
5149         \tl_set:Nn \l_tmpa_tl { #####1 }
5150         \tl_if_in:NnTF \l_tmpa_tl { - }
5151             { \@@_cut_on_hyphen:w #####1 \q_stop }
5152             { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5153         \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5154         \tl_if_empty:NT \l_tmpb_tl
5155             { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5156         \int_compare:nNnT \l_tmpb_tl > \c@iRow
5157             { \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`.

```

5158         \seq_if_in:NxF \l_@@_corners_cells_seq
5159         { \l_tmpa_tl - \l_@@_tmpc_tl }
5160         {
5161             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5162             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5163             \@@_qpoint:n { row - \l_tmpa_tl }
5164             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5165             \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5166             \pgfpathrectanglecorners
5167                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5168                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5169         }
5170     }
5171 }
5172 }

```

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`).

```

5173 \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.

```

5174 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5175   {
5176     \clist_set_eq:NN \l_tmpa_clist #1
5177     \clist_clear:N #1
5178     \clist_map_inline:Nn \l_tmpa_clist
5179     {
5180       \tl_set:Nn \l_tmpa_tl { ##1 }
5181       \tl_if_in:NnTF \l_tmpa_tl { - }
5182         { \@@_cut_on_hyphen:w ##1 \q_stop }
5183         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5184       \bool_lazy_or:nnT
5185         { \tl_if_blank_p:V \l_tmpa_tl }
5186         { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5187         { \tl_set:Nn \l_tmpa_tl { 1 } }
5188       \bool_lazy_or:nnT
5189         { \tl_if_blank_p:V \l_tmpb_tl }
5190         { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5191         { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5192       \int_compare:nNnT \l_tmpb_tl > #2
5193         { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5194       \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5195         { \clist_put_right:Nn #1 { #####1 } }
5196     }
5197   }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the `tabular`.

```

5198 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5199   {
5200     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5201     {

```

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`).

```

5202       \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5203       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5204     }
5205     \ignorespaces
5206   }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the `tabular`.

```

5207 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5208   {
5209     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5210     {
5211       \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5212       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5213       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5214     }
5215     \ignorespaces
5216   }

```

```

5217 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5218   {

```

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).

```

5219     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5220     {

```

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).

```

5221     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5222     {
5223         \exp_not:N \columncolor [ #1 ]
5224         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5225     }
5226 }
5227 }
```

## The vertical and horizontal rules

### OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumn` type 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`.

```

5228 \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`.

```

5229 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5230 {
5231     \int_compare:nNnTF \l_@@_first_col_int = 0
5232     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5233     {
5234         \int_compare:nNnTF \c@jCol = 0
5235         {
5236             \int_compare:nNnF \c@iRow = { -1 }
5237             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5238         }
5239         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5240     }
5241 }
```

This definition may seem complicated but we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell.

The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```

5242 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5243 {
5244     \int_compare:nNnF \c@iRow = 0
5245     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
5246 }
```

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



## General system for drawing rules

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

```
5247 \keys_define:nn { NiceMatrix / Rules }
5248   {
5249     position .int_set:N = \l_@@_position_int ,
5250     position .value_required:n = true ,
5251     start .int_set:N = \l_@@_start_int ,
5252     start .initial:n = 1 ,
5253     end .code:n =
5254       \bool_lazy_or:nnTF
5255         { \tl_if_empty_p:n { #1 } }
5256         { \str_if_eq_p:nn { #1 } { last } }
5257         { \int_set_eq:NN \l_@@_end_int \c@jCol }
5258         { \int_set:Nn \l_@@_end_int { #1 } }
5259   }
```

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

```
5260 \keys_define:nn { NiceMatrix / RulesBis }
5261   {
5262     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5263     multiplicity .initial:n = 1 ,
5264     dotted .bool_set:N = \l_@@_dotted_bool ,
5265     dotted .initial:n = false ,
5266     dotted .default:n = true ,
5267     color .code:n = \@@_set_CT@arc@:n { #1 } ,
5268     color .value_required:n = true ,
5269     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5270     sep-color .value_required:n = true ,
```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with `Tikz`.

```
5271     tikz .tl_set:N = \l_@@_tikz_rule_tl ,
5272     tikz .value_required:n = true ,
5273     tikz .initial:n = ,
5274     total-width .dim_set:N = \l_@@_rule_width_dim ,
5275     total-width .value_required:n = true ,
5276     width .meta:n = { total-width = #1 } ,
5277     unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
5278   }
```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```
5279 \cs_new_protected:Npn \@@_vline:n #1
5280   {
```

The group is for the options.

```
5281     \group_begin:
5282     \int_zero_new:N \l_@@_end_int
5283     \int_set_eq:NN \l_@@_end_int \c@iRow
5284     \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of |c|c|c| but only two columns used).

```

5285   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5286     \@@_vline_i:
5287   \group_end:
5288 }

5289 \cs_new_protected:Npn \@@_vline_i:
5290 {
5291   \int_zero_new:N \l_@@_local_start_int
5292   \int_zero_new:N \l_@@_local_end_int

```

\l\_tmpa\_tl is the number of row and \l\_tmpb\_tl the number of column. When we have found a row corresponding to a rule to draw, we note its number in \l\_@@\_tmpc\_tl.

```

5293   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5294   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5295     \l_tmpa_tl
5296   {

```

The boolean \g\_tmpa\_bool indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by \Block or a virtual block corresponding to a dotted line, created by \Cdots, \Vdots, etc.), we will set \g\_tmpa\_bool to false and the small vertical rule won't be drawn.

```

5297     \bool_gset_true:N \g_tmpa_bool
5298     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5299       { \@@_test_vline_in_block:nnnnn ##1 }
5300     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5301       { \@@_test_vline_in_block:nnnnn ##1 }
5302     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5303       { \@@_test_vline_in_stroken_block:nnnn ##1 }
5304     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5305     \bool_if:NTF \g_tmpa_bool
5306       {
5307         \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. \l\_@@\_local\_start\_int will be the starting row of the rule that we will have to draw.

```

5308         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5309       }
5310     {
5311       \int_compare:nNnT \l_@@_local_start_int > 0
5312       {
5313         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5314         \@@_vline_ii:
5315         \int_zero:N \l_@@_local_start_int
5316       }
5317     }
5318   }
5319   \int_compare:nNnT \l_@@_local_start_int > 0
5320   {
5321     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5322     \@@_vline_ii:
5323   }
5324 }

```

```

5325 \cs_new_protected:Npn \@@_test_in_corner_v:
5326 {
5327   \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5328   {
5329     \seq_if_in:NxT
5330     \l_@@_corners_cells_seq
5331     { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5332     { \bool_set_false:N \g_tmpa_bool }
5333   }

```

```

5334 {
5335   \seq_if_in:NxT
5336   \l_@@_corners_cells_seq
5337   { \l_tmpa_tl - \l_tmpb_tl }
5338   {
5339     \int_compare:nNnTF \l_tmpb_tl = 1
5340     { \bool_set_false:N \g_tmpa_bool }
5341     {
5342       \seq_if_in:NxT
5343       \l_@@_corners_cells_seq
5344       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5345       { \bool_set_false:N \g_tmpa_bool }
5346     }
5347   }
5348 }
5349 }

```

```

5350 \cs_new_protected:Npn \@@_vline_ii:
5351 {
5352   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5353   \bool_if:NTF \l_@@_dotted_bool
5354   \@@_vline_iv:
5355   {
5356     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5357     \@@_vline_iii:
5358     \@@_vline_v:
5359   }
5360 }

```

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

```

5361 \cs_new_protected:Npn \@@_vline_iii:
5362 {
5363   \pgfpicture
5364   \pgfrememberpicturepositiononpagetrue
5365   \pgf@relevantforpicturesizefalse
5366   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5367   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5368   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5369   \dim_set:Nn \l_tmpb_dim
5370   {
5371     \pgf@x
5372     - 0.5 \l_@@_rule_width_dim
5373     +
5374     ( \arrayrulewidth * \l_@@_multiplicity_int
5375       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5376   }
5377   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5378   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5379   \bool_lazy_all:nT
5380   {
5381     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5382     { \cs_if_exist_p:N \CT@drsc@ }
5383     { ! \tl_if_blank_p:V \CT@drsc@ }
5384   }
5385   {
5386     \group_begin:
5387     \CT@drsc@
5388     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5389     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5390     \dim_set:Nn \l_@@_tmpd_dim
5391     {
5392       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )

```

```

5393         * ( \l_@@_multiplicity_int - 1 )
5394     }
5395     \pgfpathrectanglecorners
5396     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5397     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5398     \pgfusepath { fill }
5399     \group_end:
5400 }
5401 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5402 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5403 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5404 {
5405     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5406     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5407     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5408     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5409 }
5410 \CT@arc@
5411 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5412 \pgfsetrectcap
5413 \pgfusepathqstroke
5414 \endpgfpicture
5415 }

```

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

```

5416 \cs_new_protected:Npn \@@_vline_iv:
5417 {
5418     \pgfpicture
5419     \pgfrememberpicturepositiononpagetrue
5420     \pgf@relevantforpicturesizefalse
5421     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5422     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5423     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5424     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5425     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5426     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5427     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5428     \CT@arc@
5429     \@@_draw_line:
5430     \endpgfpicture
5431 }

```

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

```

5432 \cs_new_protected:Npn \@@_vline_v:
5433 {
5434     \begin {tikzpicture }
5435     \pgfrememberpicturepositiononpagetrue
5436     \pgf@relevantforpicturesizefalse
5437     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5438     \dim_set_eq:NN \l_tmpa_dim \pgf@y
5439     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5440     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5441     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5442     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5443     \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5444     \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5445     ( \l_tmpb_dim , \l_tmpa_dim ) --
5446     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5447     \end {tikzpicture }
5448 }

```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

5449 \cs_new_protected:Npn \@@_draw_vlines:
5450   {
5451     \int_step_inline:nnn
5452       {
5453         \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5454           1 2
5455       }
5456     {
5457       \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5458         { \int_eval:n { \c@jCol + 1 } }
5459         \c@jCol
5460     }
5461   {
5462     \tl_if_eq:NnF \l_@@_vlines_clist { all }
5463     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5464     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5465   }
5466 }

```

## The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs of the form `{NiceMatrix/Rules}`.

```

5467 \cs_new_protected:Npn \@@_hline:n #1
5468   {

```

The group is for the options.

```

5469   \group_begin:
5470   \int_zero_new:N \l_@@_end_int
5471   \int_set_eq:NN \l_@@_end_int \c@jCol
5472   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5473   \@@_hline_i:
5474   \group_end:
5475 }

```

```

5476 \cs_new_protected:Npn \@@_hline_i:

```

```

5477   {
5478     \int_zero_new:N \l_@@_local_start_int
5479     \int_zero_new:N \l_@@_local_end_int

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a column corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

5480   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5481   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5482     \l_tmpb_tl
5483     {

```

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.

```

5484     \bool_gset_true:N \g_tmpa_bool
5485     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5486       { \@@_test_hline_in_block:nnnn ##1 }
5487     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5488       { \@@_test_hline_in_block:nnnn ##1 }
5489     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5490       { \@@_test_hline_in_stroken_block:nnnn ##1 }
5491     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5492     \bool_if:NTF \g_tmpa_bool
5493       {
5494         \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```

5495         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5496     }
5497     {
5498         \int_compare:nNnT \l_@@_local_start_int > 0
5499         {
5500             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5501             \@@_hline_ii:
5502             \int_zero:N \l_@@_local_start_int
5503         }
5504     }
5505 }
5506 \int_compare:nNnT \l_@@_local_start_int > 0
5507 {
5508     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5509     \@@_hline_ii:
5510 }
5511 }

```

```

5512 \cs_new_protected:Npn \@@_test_in_corner_h:
5513 {
5514     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5515     {
5516         \seq_if_in:NxT
5517         \l_@@_corners_cells_seq
5518         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5519         { \bool_set_false:N \g_tmpa_bool }
5520     }
5521     {
5522         \seq_if_in:NxT
5523         \l_@@_corners_cells_seq
5524         { \l_tmpa_tl - \l_tmpb_tl }
5525         {
5526             \int_compare:nNnTF \l_tmpa_tl = 1
5527             { \bool_set_false:N \g_tmpa_bool }
5528             {
5529                 \seq_if_in:NxT
5530                 \l_@@_corners_cells_seq
5531                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5532                 { \bool_set_false:N \g_tmpa_bool }
5533             }
5534         }
5535     }
5536 }

```

```

5537 \cs_new_protected:Npn \@@_hline_ii:
5538 {
5539     % \bool_set_false:N \l_@@_dotted_bool
5540     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5541     \bool_if:NTF \l_@@_dotted_bool
5542     \@@_hline_iv:
5543     {
5544         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5545         \@@_hline_iii:
5546         \@@_hline_v:
5547     }
5548 }

```

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

```

5549 \cs_new_protected:Npn \@@_hline_iii:

```

```

5550 {
5551   \pgfpicture
5552   \pgfrememberpicturepositiononpagetrue
5553   \pgf@relevantforpicturesizefalse
5554   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5555   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5556   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5557   \dim_set:Nn \l_tmpb_dim
5558     {
5559       \pgf@y
5560       - 0.5 \l_@@_rule_width_dim
5561       +
5562       ( \arrayrulewidth * \l_@@_multiplicity_int
5563         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5564     }
5565   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5566   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5567   \bool_lazy_all:nT
5568     {
5569       { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5570       { \cs_if_exist_p:N \CT@drsc@ }
5571       { ! \tl_if_blank_p:V \CT@drsc@ }
5572     }
5573     {
5574       \group_begin:
5575       \CT@drsc@
5576       \dim_set:Nn \l_@@_tmpd_dim
5577         {
5578           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5579           * ( \l_@@_multiplicity_int - 1 )
5580         }
5581       \pgfpathrectanglecorners
5582         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5583         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5584       \pgfusepathqfill
5585       \group_end:
5586     }
5587   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5588   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5589   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5590     {
5591       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5592       \dim_sub:Nn \l_tmpb_dim \doublerulesep
5593       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5594       \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5595     }
5596   \CT@arc@
5597   \pgfsetlinewidth { 1.1 \arrayrulewidth }
5598   \pgfsetrectcap
5599   \pgfusepathqstroke
5600   \endpgfpicture
5601 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

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}

```

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

```

5602 \cs_new_protected:Npn \@@_hline_iv:
5603   {
5604     \pgfpicture
5605     \pgfrememberpicturepositiononpagetrue
5606     \pgf@relevantforpicturesizefalse
5607     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5608     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5609     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
5610     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5611     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5612     \int_compare:nNnT \l_@@_local_start_int = 1
5613     {
5614       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5615       \bool_if:NT \g_@@_NiceArray_bool
5616       { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by  $0.5 \l_@@_xdots\_inter\_dim$  is *ad hoc* for a better result.

```

5617     \tl_if_eq:NnF \g_@@_left_delim_tl (
5618     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
5619     )
5620     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5621     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5622     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5623     {
5624       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5625       \bool_if:NT \g_@@_NiceArray_bool
5626       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5627       \tl_if_eq:NnF \g_@@_right_delim_tl )
5628       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
5629     }
5630     \CT@arc@
5631     \@@_draw_line:
5632     \endpgfpicture
5633   }

```

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

```

5634 \cs_new_protected:Npn \@@_hline_v:
5635   {
5636     \begin { tikzpicture }
5637     \pgfrememberpicturepositiononpagetrue
5638     \pgf@relevantforpicturesizefalse
5639     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5640     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5641     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5642     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5643     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5644     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5645     \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5646     \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5647     ( \l_tmpa_dim , \l_tmpb_dim ) --
5648     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5649     \end { tikzpicture }
5650   }

```



The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners (if the key `corners` is used)).

```

5651 \cs_new_protected:Npn \@@_draw_hlines:
5652   {
5653     \int_step_inline:nnn
5654       {
5655         \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5656           1 2
5657       }
5658     {
5659       \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5660         { \int_eval:n { \c@iRow + 1 } }
5661         \c@iRow
5662     }
5663   {
5664     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5665     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5666     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
5667   }
5668 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5669 \cs_set:Npn \@@_Hline: { \noalign \bgroup \@@_Hline_i:n { 1 } }

```

The argument of the command `\@@_Hline_i:n` is the number of successive `\Hline` found.

```

5670 \cs_set:Npn \@@_Hline_i:n #1
5671   {
5672     \peek_remove_spaces:n
5673     {
5674       \peek_meaning:NTF \Hline
5675         { \@@_Hline_ii:nn { #1 + 1 } }
5676         { \@@_Hline_iii:n { #1 } }
5677     }
5678 }
5679 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5680 \cs_set:Npn \@@_Hline_iii:n #1
5681   {
5682     \peek_meaning:NTF [
5683       { \@@_Hline_iv:nw { #1 } }
5684       { \@@_Hline_iv:nw { #1 } [ ] }
5685     ]
5686 \cs_set:Npn \@@_Hline_iv:nw #1 [ #2 ]
5687   {
5688     \@@_compute_rule_width:n { multiplicity = #1 , #2 }
5689     \skip_vertical:n { \l_@@_rule_width_dim }
5690     \tl_gput_right:Nx \g_@@_pre_code_after_tl
5691     {
5692       \@@_hline:n
5693       {
5694         multiplicity = #1 ,
5695         position = \int_eval:n { \c@iRow + 1 } ,
5696         total-width = \dim_use:N \l_@@_rule_width_dim ,
5697         #2
5698       }
5699     }
5700     \egroup
5701   }

```

## Customized rules defined by the final user

The final user can define a customized rule by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of `key=value` pairs.

Among the keys available in that list, there is the key `letter` to specify a letter that the final user will use in the preamble of the array. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix / ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```
5702 \keys_define:nn { NiceMatrix / ColumnTypes } { }
```

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```
5703 \cs_new_protected:Npn \@@_custom_line:n #1
5704 {
5705   \str_clear_new:N \l_@@_command_str
5706   \str_clear_new:N \l_@@_ccommand_str
5707   \str_clear_new:N \l_@@_letter_str
5708   \keys_set_known:nnN { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl
```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```
5709   \bool_lazy_all:nTF
5710     {
5711       { \str_if_empty_p:N \l_@@_letter_str }
5712       { \str_if_empty_p:N \l_@@_command_str }
5713       { \str_if_empty_p:N \l_@@_ccommand_str }
5714     }
5715     { \@@_error:n { No-letter~and~no-command } }
5716     { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
5717   }

5718 \keys_define:nn { NiceMatrix / custom-line }
5719 {
5720   % here, we will use change in the future to use .str_set:N
5721   letter .code:n = \str_set:Nn \l_@@_letter_str { #1 } ,
5722   letter .value_required:n = true ,
5723   command .code:n = \str_set:Nn \l_@@_command_str { #1 } ,
5724   command .value_required:n = true ,
5725   ccommand .code:n = \str_set:Nn \l_@@_ccommand_str { #1 } ,
5726   ccommand .value_required:n = true ,
5727 }
```

```
5728 \cs_new_protected:Npn \@@_custom_line_i:n #1
5729 {
```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```
5730   \bool_set_false:N \l_@@_tikz_rule_bool
5731   \bool_set_false:N \l_@@_dotted_rule_bool
5732   \bool_set_false:N \l_@@_color_bool

5733   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
5734   \bool_if:NT \l_@@_tikz_rule_bool
5735     {
```

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.

```
5736     \cs_if_exist:NF \tikzpicture
5737     { \@@_error:n { tikz~in~custom-line~without~tikz } }
```

```

5738     \bool_if:NT \l_@@_color_bool
5739     { \@@_error:n { color~in~custom~line~with~tikz } }
5740   }
5741   \bool_if:nT
5742   {
5743     \int_compare_p:nNn \l_@@_multiplicity_int > 1
5744     && \l_@@_dotted_rule_bool
5745   }
5746   { \@@_error:n { key~multiplicity~with~dotted } }
5747   \str_if_empty:NF \l_@@_letter_str
5748   {
5749     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
5750     { \@@_error:n { Several~letters } }
5751     {
5752       \exp_args:NnV \tl_if_in:NnTF
5753       \c_@@_forbidden_letters_str \l_@@_letter_str
5754       { \@@_error:n { Forbidden~letter } }
5755       {

```

The final user can, locally, redefine a letter of column type. That's compatible with the use of `\keys_define:nn`: the definition is local and may overwrite a previous definition.

```

5756         \keys_define:nx { NiceMatrix / ColumnTypes }
5757         {
5758           \l_@@_letter_str .code:n =
5759           { \@@_v_custom_line:n { \exp_not:n { #1 } } }
5760         }
5761       }
5762     }
5763   }
5764   \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
5765   \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
5766 }
5767 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

5768 \keys_define:nn { NiceMatrix / custom-line-bis }
5769 {
5770   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5771   multiplicity .initial:n = 1 ,
5772   multiplicity .value_required:n = true ,
5773   color .code:n = \bool_set_true:N \l_@@_color_bool ,
5774   color .value_required:n = true ,
5775   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5776   tikz .value_required:n = true ,
5777   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5778   dotted .value_forbidden:n = true ,
5779   total-width .code:n = { } ,
5780   total-width .value_required:n = true ,
5781   width .code:n = { } ,
5782   width .value_required:n = true ,
5783   sep-color .code:n = { } ,
5784   sep-color .value_required:n = true ,
5785   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
5786 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

5787 \bool_new:N \l_@@_dotted_rule_bool
5788 \bool_new:N \l_@@_tikz_rule_bool
5789 \bool_new:N \l_@@_color_bool

```

The following keys are used to determine the total width of the line (including the spaces on both sides of the line). The key `width` is deprecated and has been replaced by the key `total-width`.

```

5790 \keys_define:nn { NiceMatrix / custom-line-width }
5791   {
5792     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5793     multiplicity .initial:n = 1 ,
5794     multiplicity .value_required:n = true ,
5795     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5796     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5797                           \bool_set_true:N \l_@@_total_width_bool ,
5798     total-width .value_required:n = true ,
5799     width .meta:n = { total-width = #1 } ,
5800     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5801   }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

5802 \cs_new_protected:Npn \@@_h_custom_line:n #1
5803   {

```

We use `\cs_set:cpn` and not `\cs_new:cpn` because we want a local definition. Moreover, the command must *not* be protected since it begins with `\noalign`.

```

5804     \cs_set:cpn { nicematrix - \l_@@_command_str }
5805     {
5806       \noalign
5807       {
5808         \@@_compute_rule_width:n { #1 }
5809         \skip_vertical:n { \l_@@_rule_width_dim }
5810         \tl_gput_right:Nx \g_@@_pre_code_after_tl
5811         {
5812           \@@_hline:n
5813           {
5814             #1 ,
5815             position = \int_eval:n { \c@iRow + 1 } ,
5816             total-width = \dim_use:N \l_@@_rule_width_dim
5817           }
5818         }
5819       }
5820     }
5821     \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
5822   }
5823 \cs_generate_variant:Nn \@@_h_custom_line:nn { n V }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter `c` as in `\cline`). `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

5824 \cs_new_protected:Npn \@@_c_custom_line:n #1
5825   {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

5826     \exp_args:Nc \NewExpandableDocumentCommand
5827     { nicematrix - \l_@@_ccommand_str }
5828     { 0 { } m }
5829     {
5830       \noalign
5831       {
5832         \@@_compute_rule_width:n { #1 , ##1 }
5833         \skip_vertical:n { \l_@@_rule_width_dim }
5834         \clist_map_inline:nn
5835         { ##2 }

```

```

5836         { \l_@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
5837     }
5838 }
5839 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
5840 }

```

The first argument is the list of key-value pairs characteristic of the line. The second argument is the specification of columns for the `\cline` with the syntax *a-b*.

```

5841 \cs_new_protected:Npn \l_@@_c_custom_line_i:nn #1 #2
5842 {
5843   \str_if_in:nnTF { #2 } { - }
5844   { \l_@@_cut_on_hyphen:w #2 \q_stop }
5845   { \l_@@_cut_on_hyphen:w #2 - #2 \q_stop }
5846   \tl_gput_right:Nx \g_@@_pre_code_after_tl
5847   {
5848     \l_@@_hline:n
5849     {
5850       #1 ,
5851       start = \l_tmpa_tl ,
5852       end = \l_tmpb_tl ,
5853       position = \int_eval:n { \c@iRow + 1 } ,
5854       total-width = \dim_use:N \l_@@_rule_width_dim
5855     }
5856   }
5857 }
5858 \cs_generate_variant:Nn \l_@@_c_custom_line:nn { n V }
5859 \cs_new_protected:Npn \l_@@_compute_rule_width:n #1
5860 {
5861   \bool_set_false:N \l_@@_tikz_rule_bool
5862   \bool_set_false:N \l_@@_total_width_bool
5863   \bool_set_false:N \l_@@_dotted_rule_bool
5864   \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
5865   \bool_if:NF \l_@@_total_width_bool
5866   {
5867     \bool_if:NTF \l_@@_dotted_rule_bool
5868     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
5869     {
5870       \bool_if:NF \l_@@_tikz_rule_bool
5871       {
5872         \dim_set:Nn \l_@@_rule_width_dim
5873         {
5874           \arrayrulewidth * \l_@@_multiplicity_int
5875           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
5876         }
5877       }
5878     }
5879   }
5880 }
5881 \cs_new_protected:Npn \l_@@_v_custom_line:n #1
5882 {
5883   \l_@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

5884   \tl_gput_right:Nx \g_@@_preamble_tl
5885   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
5886   \tl_gput_right:Nx \g_@@_pre_code_after_tl
5887   {
5888     \l_@@_vline:n
5889     {
5890       #1 ,
5891       position = \int_eval:n { \c@jCol + 1 } ,
5892       total-width = \dim_use:N \l_@@_rule_width_dim
5893     }

```

```

5894     }
5895   }
5896 \@@_custom_line:n
5897   { letter = : , command = hdottedline , ccommand = cdottedline, dotted }

```

## The key hvlines

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

```

5898 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
5899   {
5900     \bool_lazy_all:nT
5901     {
5902       { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
5903       { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5904       { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5905       { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5906     }
5907     { \bool_gset_false:N \g_tmpa_bool }
5908   }

```

The same for vertical rules.

```

5909 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
5910   {
5911     \bool_lazy_all:nT
5912     {
5913       { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5914       { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5915       { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
5916       { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5917     }
5918     { \bool_gset_false:N \g_tmpa_bool }
5919   }
5920 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
5921   {
5922     \bool_lazy_all:nT
5923     {
5924       {
5925         ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
5926         || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
5927       }
5928       { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5929       { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5930     }
5931     { \bool_gset_false:N \g_tmpa_bool }
5932   }
5933 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
5934   {
5935     \bool_lazy_all:nT
5936     {
5937       { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5938       { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5939       {
5940         ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
5941         || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
5942       }
5943     }
5944     { \bool_gset_false:N \g_tmpa_bool }
5945   }

```

## 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.

```
5946 \cs_new_protected:Npn \@@_compute_corners:
5947 {
```

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.

```
5948   \seq_clear_new:N \l_@@_corners_cells_seq
5949   \clist_map_inline:Nn \l_@@_corners_clist
5950     {
5951     \str_case:nnF { ##1 }
5952       {
5953         { NW }
5954         { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
5955         { NE }
5956         { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
5957         { SW }
5958         { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
5959         { SE }
5960         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
5961       }
5962     { \@@_error:nn { bad~corner } { ##1 } }
5963   }
```

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

```
5964   \seq_if_empty:NF \l_@@_corners_cells_seq
5965   {
```

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.

```
5966     \tl_gput_right:Nx \g_@@_aux_tl
5967     {
5968       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
5969       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
5970     }
5971   }
5972 }
```

“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.

```
5973 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
5974 {
```

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.

```
5975   \bool_set_false:N \l_tmpa_bool
5976   \int_zero_new:N \l_@@_last_empty_row_int
5977   \int_set:Nn \l_@@_last_empty_row_int { #1 }
```

```

5978 \int_step_inline:nnnn { #1 } { #3 } { #5 }
5979 {
5980   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
5981   \bool_lazy_or:nnTF
5982   {
5983     \cs_if_exist_p:c
5984     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
5985   }
5986   \l_tmpb_bool
5987   { \bool_set_true:N \l_tmpa_bool }
5988   {
5989     \bool_if:NF \l_tmpa_bool
5990     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
5991   }
5992 }

```

Now, you determine the last empty cell in the row of number 1.

```

5993 \bool_set_false:N \l_tmpa_bool
5994 \int_zero_new:N \l_@@_last_empty_column_int
5995 \int_set:Nn \l_@@_last_empty_column_int { #2 }
5996 \int_step_inline:nnnn { #2 } { #4 } { #6 }
5997 {
5998   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
5999   \bool_lazy_or:nnTF
6000   \l_tmpb_bool
6001   {
6002     \cs_if_exist_p:c
6003     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6004   }
6005   { \bool_set_true:N \l_tmpa_bool }
6006   {
6007     \bool_if:NF \l_tmpa_bool
6008     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6009   }
6010 }

```

Now, we loop over the rows.

```

6011 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6012 {

```

We treat the row number ##1 with another loop.

```

6013   \bool_set_false:N \l_tmpa_bool
6014   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6015   {
6016     \@@_test_if_cell_in_a_block:nn { ##1 } { ##### }
6017     \bool_lazy_or:nnTF
6018     \l_tmpb_bool
6019     {
6020       \cs_if_exist_p:c
6021       { pgf @ sh @ ns @ \@@_env: - ##1 - ##### }
6022     }
6023     { \bool_set_true:N \l_tmpa_bool }
6024     {
6025       \bool_if:NF \l_tmpa_bool
6026       {
6027         \int_set:Nn \l_@@_last_empty_column_int { ##### }
6028         \seq_put_right:Nn
6029         \l_@@_corners_cells_seq
6030         { ##1 - ##### }
6031       }
6032     }
6033   }
6034 }
6035 }

```



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`).

```

6036 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6037   {
6038     \int_set:Nn \l_tmpa_int { #1 }
6039     \int_set:Nn \l_tmpb_int { #2 }
6040     \bool_set_false:N \l_tmpb_bool
6041     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6042       { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6043   }
6044 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6 #7
6045   {
6046     \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6047     {
6048       \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6049       {
6050         \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6051         {
6052           \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6053           { \bool_set_true:N \l_tmpb_bool }
6054         }
6055       }
6056     }
6057   }

```

## 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.

```

6058 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6059 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6060   {
6061     auto-columns-width .code:n =
6062     {
6063       \bool_set_true:N \l_@@_block_auto_columns_width_bool
6064       \dim_gzero_new:N \g_@@_max_cell_width_dim
6065       \bool_set_true:N \l_@@_auto_columns_width_bool
6066     }
6067   }
6068 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6069   {
6070     \int_gincr:N \g_@@_NiceMatrixBlock_int
6071     \dim_zero:N \l_@@_columns_width_dim
6072     \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6073     \bool_if:NT \l_@@_block_auto_columns_width_bool
6074     {
6075       \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6076       {
6077         \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
6078           { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6079       }
6080     }
6081   }

```

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`).

```

6082 {
6083   \bool_if:NT \l_@@_block_auto_columns_width_bool
6084   {
6085     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6086     \iow_shipout:Nx \@mainaux
6087     {
6088       \cs_gset:cpn
6089       { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6090       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6091     }
6092     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6093   }
6094 }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

## The extra nodes

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

```

6095 \cs_generate_variant:Nn \dim_min:nn { v n }
6096 \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).

```

6097 \cs_new_protected:Npn \@@_create_extra_nodes:
6098 {
6099   \bool_if:nTF \l_@@_medium_nodes_bool
6100   {
6101     \bool_if:NTF \l_@@_large_nodes_bool
6102     \@@_create_medium_and_large_nodes:
6103     \@@_create_medium_nodes:
6104   }
6105   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6106 }

```

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`.

```

6107 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6108 {
6109   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:

```

```

6110 {
6111   \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6112   \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6113   \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6114   \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6115 }
6116 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6117 {
6118   \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6119   \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6120   \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6121   \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6122 }

```

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

```

6123 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6124 {
6125   \int_step_variable:nnNn
6126   \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.

```

6127 {
6128   \cs_if_exist:cT
6129   { 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`.

```

6130 {
6131   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6132   \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6133   { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6134   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6135   {
6136     \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6137     { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6138   }

```

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`.

```

6139   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6140   \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6141   { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6142   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6143   {
6144     \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6145     { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6146   }
6147 }
6148 }
6149 }

```

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

```

6150 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6151 {
6152   \dim_compare:nNnT
6153   { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6154   {
6155     \@@_qpoint:n { row - \@@_i: - base }
6156     \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6157     \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6158   }
6159 }
6160 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

```

6161     {
6162       \dim_compare:nNnT
6163         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6164         {
6165           \@@_qpoint:n { col - \@@_j: }
6166           \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6167           \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6168         }
6169     }
6170 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6171 \cs_new_protected:Npn \@@_create_medium_nodes:
6172 {
6173   \pgfpicture
6174   \pgfrememberpicturepositiononpagetrue
6175   \pgf@relevantforpicturesizefalse
6176   \@@_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”.

```

6177   \tl_set:Nn \l_@@_suffix_tl { -medium }
6178   \@@_create_nodes:
6179   \endpgfpicture
6180 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones<sup>79</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:`.

```

6181 \cs_new_protected:Npn \@@_create_large_nodes:
6182 {
6183   \pgfpicture
6184   \pgfrememberpicturepositiononpagetrue
6185   \pgf@relevantforpicturesizefalse
6186   \@@_computations_for_medium_nodes:
6187   \@@_computations_for_large_nodes:
6188   \tl_set:Nn \l_@@_suffix_tl { -large }
6189   \@@_create_nodes:
6190   \endpgfpicture
6191 }
6192 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6193 {
6194   \pgfpicture
6195   \pgfrememberpicturepositiononpagetrue
6196   \pgf@relevantforpicturesizefalse
6197   \@@_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”.

```

6198   \tl_set:Nn \l_@@_suffix_tl { -medium }
6199   \@@_create_nodes:
6200   \@@_computations_for_large_nodes:
6201   \tl_set:Nn \l_@@_suffix_tl { -large }
6202   \@@_create_nodes:
6203   \endpgfpicture
6204 }

```

---

<sup>79</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

For “large nodes”, the exterior rows and columns don’t interfer. 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.

```
6205 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6206 {
6207   \int_set:Nn \l_@@_first_row_int 1
6208   \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`.

```
6209   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6210   {
6211     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6212     {
6213       (
6214         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6215         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6216       )
6217       / 2
6218     }
6219     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6220     { l_@@_row _ \@@_i: _ min _ dim }
6221   }
6222   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6223   {
6224     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6225     {
6226       (
6227         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6228         \dim_use:c
6229         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6230       )
6231       / 2
6232     }
6233     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6234     { l_@@_column _ \@@_j: _ max _ dim }
6235   }
```

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

```
6236   \dim_sub:cn
6237   { l_@@_column _ 1 _ min _ dim }
6238   \l_@@_left_margin_dim
6239   \dim_add:cn
6240   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6241   \l_@@_right_margin_dim
6242 }
```

The command `\@@_create_nodes:` is used twice: for the construction of the “medium nodes” and for the construction of the “large nodes”. The nodes are constructed with the value of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. Between the construction of the “medium nodes” and the “large nodes”, the values of these dimensions are changed.

The function also uses `\l_@@_suffix_tl` (-medium or -large).

```
6243 \cs_new_protected:Npn \@@_create_nodes:
6244 {
6245   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6246   {
6247     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6248     {
```

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

```
6249       \@@_pgf_rect_node:nnnnn
6250       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6251       { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
```

```

6252     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6253     { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6254     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6255     \str_if_empty:NF \l_@@_name_str
6256     {
6257         \pgfnodealias
6258         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6259         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6260     }
6261 }
6262 }

```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{n}{...}{...}` with  $n > 1$  was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of  $n$ .

```

6263     \seq_mapthread_function:NNN
6264     \g_@@_multicolumn_cells_seq
6265     \g_@@_multicolumn_sizes_seq
6266     \@@_node_for_multicolumn:nn
6267 }

6268 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6269 {
6270     \cs_set_nopar:Npn \@@_i: { #1 }
6271     \cs_set_nopar:Npn \@@_j: { #2 }
6272 }

```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{n}{...}{...}` was issued in the format  $i-j$  and the second is the value of  $n$  (the length of the “multi-cell”).

```

6273 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6274 {
6275     \@@_extract_coords_values: #1 \q_stop
6276     \@@_pgf_rect_node:nnnnn
6277     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6278     { \dim_use:c { l_@@_column_ \@@_j: _min _dim } }
6279     { \dim_use:c { l_@@_row_ \@@_i: _min _dim } }
6280     { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max _dim } }
6281     { \dim_use:c { l_@@_row_ \@@_i: _max _dim } }
6282     \str_if_empty:NF \l_@@_name_str
6283     {
6284         \pgfnodealias
6285         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6286         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6287     }
6288 }

```

## 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.

```

6289 \keys_define:nn { NiceMatrix / Block / FirstPass }
6290 {
6291     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6292     l .value_forbidden:n = true ,
6293     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6294     r .value_forbidden:n = true ,

```

```

6295   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6296   c .value_forbidden:n = true ,
6297   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6298   L .value_forbidden:n = true ,
6299   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6300   R .value_forbidden:n = true ,
6301   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6302   C .value_forbidden:n = true ,
6303   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6304   t .value_forbidden:n = true ,
6305   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6306   b .value_forbidden:n = true ,
6307   color .tl_set:N = \l_@@_color_tl ,
6308   color .value_required:n = true ,
6309   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6310   respect-arraystretch .default:n = true ,
6311 }

```

The following command `\@@_Block:` will be linked to `\Block` in the environments of `nicematrix`. We define it with `\NewExpandableDocumentCommand` because it has an optional argument between `<` and `>`. It's mandatory to use an expandable command.

```

6312 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
6313 {

```

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).

```

6314   \peek_remove_spaces:n
6315   {
6316     \tl_if_blank:nTF { #2 }
6317     { \@@_Block_i 1-1 \q_stop }
6318     {
6319       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6320       \@@_Block_i_czech \@@_Block_i
6321       #2 \q_stop
6322     }
6323     { #1 } { #3 } { #4 }
6324   }
6325 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

6326 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

With `babel` with the key `czech`, the character `-` (hyphen) is active. That's why we need a special version. Remark that we could not use a preprocessor in the command `\@@_Block:` to do the job because the command `\@@_Block:` is defined with the command `\NewExpandableDocumentCommand`.

```

6327 {
6328   \char_set_catcode_active:N -
6329   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6330 }

```

Now, the arguments have been extracted: `#1` is  $i$  (the number of rows of the block), `#2` is  $j$  (the number of columns of the block), `#3` is the list of *key=values* pairs, `#4` are the tokens to put before the math mode and the beginning of the small array of the block and `#5` is the label of the block.

```

6331 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6332 {

```

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).

```

6333   \bool_lazy_or:nnTF

```

```

6334     { \tl_if_blank_p:n { #1 } }
6335     { \str_if_eq_p:nn { #1 } { * } }
6336     { \int_set:Nn \l_tmpa_int { 100 } }
6337     { \int_set:Nn \l_tmpa_int { #1 } }
6338     \bool_lazy_or:nnTF
6339     { \tl_if_blank_p:n { #2 } }
6340     { \str_if_eq_p:nn { #2 } { * } }
6341     { \int_set:Nn \l_tmpb_int { 100 } }
6342     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6343     \int_compare:nNnTF \l_tmpb_int = 1
6344     {
6345         \str_if_empty:NTF \l_@@_hpos_cell_str
6346         { \str_set:Nn \l_@@_hpos_block_str c }
6347         { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6348     }
6349     { \str_set:Nn \l_@@_hpos_block_str c }

```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```

6350     \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6351     \tl_set:Nx \l_tmpa_tl
6352     {
6353         { \int_use:N \c@iRow }
6354         { \int_use:N \c@jCol }
6355         { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6356         { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6357     }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets: `{imin}{jmin}{imax}{jmax}`.

If the block is mono-column or mono-row, we have a special treatment. That’s why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by curryfication).

```

6358     \bool_if:nTF
6359     {
6360         (
6361             \int_compare_p:nNn { \l_tmpa_int } = 1
6362             ||
6363             \int_compare_p:nNn { \l_tmpb_int } = 1
6364         )
6365         && ! \tl_if_empty_p:n { #5 }

```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

6366         && ! \l_@@_X_column_bool
6367     }
6368     { \exp_args:Nxx \@@_Block_iv:nnnnn }
6369     { \exp_args:Nxx \@@_Block_v:nnnnn }
6370     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6371 }

```

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).

```

6372 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6373 {
6374   \int_gincr:N \g_@@_block_box_int
6375   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6376   {
6377     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6378     {
6379       \@@_actually_diagbox:nnnnnn
6380       { \int_use:N \c@iRow }
6381       { \int_use:N \c@jCol }
6382       { \int_eval:n { \c@iRow + #1 - 1 } }
6383       { \int_eval:n { \c@jCol + #2 - 1 } }
6384       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6385     }
6386   }
6387   \box_gclear_new:c
6388   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
6389   \hbox_gset:cn
6390   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
6391   {

```

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with `\set@color` and not `\color_ensure_current:` (in order to use `\color_ensure_current:` safely, you should load `l3backend` before the `\documentclass` with `\RequirePackage{expl3}`).

```

6392   \tl_if_empty:NTF \l_@@_color_tl
6393   { \int_compare:nNnT { #2 } = 1 \set@color }
6394   { \@@_color:V \l_@@_color_tl }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even if it has yet been used in the beginning of the cell where the command `\Block` has been issued because we want to be able to take into account a potential instruction of color of the font in `\g_@@_row_style_tl`.

```

6395   \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
6396   \group_begin:
6397   \bool_if:NF \l_@@_respect_arraystretch_bool
6398   { \cs_set:Npn \arraystretch { 1 } }
6399   \dim_zero:N \extrarowheight
6400   #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).

```

6401   \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
6402   \bool_if:NTF \l_@@_NiceTabular_bool
6403   {
6404     \bool_lazy_all:nTF
6405     {
6406       { \int_compare_p:nNn { #2 } = 1 }
6407       { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6408       { ! \g_@@_rotate_bool }
6409     }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

6410   {
6411     \use:x
6412     {
6413       \exp_not:N \begin { minipage }%
6414       [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6415       { \l_@@_col_width_dim }
6416       \str_case:Vn \l_@@_hpos_block_str

```

```

6417         {
6418             c \centering
6419             r \raggedleft
6420             l \raggedright
6421         }
6422     }
6423     #5
6424     \end { minipage }
6425 }
6426 {
6427     \use:x
6428     {
6429         \exp_not:N \begin { tabular }%
6430         [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6431         { @ { } \l_@@_hpos_block_str @ { } }
6432     }
6433     #5
6434     \end { tabular }
6435 }
6436 }
6437 {
6438     \c_math_toggle_token
6439     \use:x
6440     {
6441         \exp_not:N \begin { array }%
6442         [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6443         { @ { } \l_@@_hpos_block_str @ { } }
6444     }
6445     #5
6446     \end { array }
6447     \c_math_toggle_token
6448 }
6449 \group_end:
6450 }
6451 \bool_if:NT \g_@@_rotate_bool
6452 {
6453     \box_grotate:cn
6454     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6455     { 90 }
6456     \bool_gset_false:N \g_@@_rotate_bool
6457 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

6458     \int_compare:nNnT { #2 } = 1
6459     {
6460         \dim_gset:Nn \g_@@_blocks_wd_dim
6461         {
6462             \dim_max:nn
6463             \g_@@_blocks_wd_dim
6464             {
6465                 \box_wd:c
6466                 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6467             }
6468         }
6469     }

```

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.

```

6470     \int_compare:nNnT { #1 } = 1
6471     {
6472         \dim_gset:Nn \g_@@_blocks_ht_dim
6473         {
6474             \dim_max:nn

```

```

6475         \g_@@_blocks_ht_dim
6476         {
6477             \box_ht:c
6478             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6479         }
6480     }
6481     \dim_gset:Nn \g_@@_blocks_dp_dim
6482     {
6483         \dim_max:nn
6484         \g_@@_blocks_dp_dim
6485         {
6486             \box_dp:c
6487             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6488         }
6489     }
6490 }
6491 \seq_gput_right:Nx \g_@@_blocks_seq
6492 {
6493     \l_tmpa_tl

```

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in `\l_@@_hpos_block_str`. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of `\l_@@_hpos_block_str`, which is fixed by the type of current column.

```

6494     { \exp_not:n { #3 } , \l_@@_hpos_block_str }
6495     {
6496         \box_use_drop:c
6497         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6498     }
6499 }
6500 }

```

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.

```

6501 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
6502 {
6503     \seq_gput_right:Nx \g_@@_blocks_seq
6504     {
6505         \l_tmpa_tl
6506         { \exp_not:n { #3 } }
6507         {
6508             \bool_if:NTF \l_@@_NiceTabular_bool
6509             {
6510                 \group_begin:
6511                 \bool_if:NF \l_@@_respect_arraystretch_bool
6512                 { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6513                 \exp_not:n
6514                 {
6515                     \dim_zero:N \extrarowheight
6516                     #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).

```

6517         \bool_if:NT \g_@@_rotate_bool
6518         { \str_set:Nn \l_@@_hpos_block_str c }
6519         \use:x
6520         {
6521             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
6522             { @ { } \l_@@_hpos_block_str @ { } }

```

```

6523         }
6524         #5
6525     \end { tabular }
6526 }
6527 \group_end:
6528 }
6529 {
6530     \group_begin:
6531     \bool_if:NF \l_@@_respect_arraystretch_bool
6532     { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
6533     \exp_not:n
6534     {
6535         \dim_zero:N \extrarowheight
6536         #4
6537         \bool_if:NT \g_@@_rotate_bool
6538         { \str_set:Nn \l_@@_hpos_block_str c }
6539         \c_math_toggle_token
6540         \use:x
6541         {
6542             \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
6543             { @ { } \l_@@_hpos_block_str @ { } }
6544         }
6545         #5
6546         \end { array }
6547         \c_math_toggle_token
6548     }
6549     \group_end:
6550 }
6551 }
6552 }
6553 }

```

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).

```

6554 \keys_define:nn { NiceMatrix / Block / SecondPass }
6555 {
6556     tikz .code:n =
6557     \bool_if:NTF \c_@@_tikz_loaded_bool
6558     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
6559     { \@@_error:n { tikz-key-without-tikz } } ,
6560     tikz .value_required:n = true ,
6561     fill .code:n =
6562     \tl_set_rescan:Nnn
6563     \l_@@_fill_tl
6564     { \char_set_catcode_other:N ! }
6565     { #1 } ,
6566     fill .value_required:n = true ,
6567     draw .code:n =
6568     \tl_set_rescan:Nnn
6569     \l_@@_draw_tl
6570     { \char_set_catcode_other:N ! }
6571     { #1 } ,
6572     draw .default:n = default ,
6573     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6574     rounded-corners .default:n = 4 pt ,
6575     color .code:n =
6576     \@@_color:n { #1 }
6577     \tl_set_rescan:Nnn
6578     \l_@@_draw_tl
6579     { \char_set_catcode_other:N ! }
6580     { #1 } ,

```

```

6581   color .value_required:n = true ,
6582   borders .clist_set:N = \l_@@_borders_clist ,
6583   borders .value_required:n = true ,
6584   hvlines .meta:n = { vlines , hlines } ,
6585   vlines .bool_set:N = \l_@@_vlines_block_bool ,
6586   vlines .default:n = true ,
6587   hlines .bool_set:N = \l_@@_hlines_block_bool ,
6588   hlines .default:n = true ,
6589   line-width .dim_set:N = \l_@@_line_width_dim ,
6590   line-width .value_required:n = true ,
6591   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6592   l .value_forbidden:n = true ,
6593   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6594   r .value_forbidden:n = true ,
6595   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6596   c .value_forbidden:n = true ,
6597   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
6598     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6599   L .value_forbidden:n = true ,
6600   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6601     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6602   R .value_forbidden:n = true ,
6603   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6604     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6605   C .value_forbidden:n = true ,
6606   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6607   t .value_forbidden:n = true ,
6608   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6609   T .value_forbidden:n = true ,
6610   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6611   b .value_forbidden:n = true ,
6612   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6613   B .value_forbidden:n = true ,
6614   v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
6615   v-center .value_forbidden:n = true ,
6616   name .tl_set:N = \l_@@_block_name_str ,
6617   name .value_required:n = true ,
6618   name .initial:n = ,
6619   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6620   respect-arraystretch .default:n = true ,
6621   transparent .bool_set:N = \l_@@_transparent_bool ,
6622   transparent .default:n = true ,
6623   transparent .initial:n = false ,
6624   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
6625 }

```

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.

```

6626 \cs_new_protected:Npn \@@_draw_blocks:
6627 {
6628   \cs_set_eq:NN \ialign \@@_old_ialign:
6629   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnn ##1 }
6630 }
6631 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5 #6
6632 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6633   \int_zero_new:N \l_@@_last_row_int
6634   \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.

```

6635 \int_compare:nNnTF { #3 } > { 99 }
6636   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
6637   { \int_set:Nn \l_@@_last_row_int { #3 } }
6638 \int_compare:nNnTF { #4 } > { 99 }
6639   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
6640   { \int_set:Nn \l_@@_last_col_int { #4 } }
6641 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6642   {
6643     \int_compare:nTF
6644       { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6645       {
6646         \msg_error:nnnn { nicematrix } { Block-too~large~2 } { #1 } { #2 }
6647         \@_msg_redirect_name:nn { Block-too~large~2 } { none }
6648         \@_msg_redirect_name:nn { columns~not~used } { none }
6649       }
6650       { \msg_error:nnnn { nicematrix } { Block-too~large~1 } { #1 } { #2 } }
6651   }
6652   {
6653     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6654       { \msg_error:nnnn { nicematrix } { Block-too~large~1 } { #1 } { #2 } }
6655       { \@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6656   }
6657 }

```

**#1** is the first row of the block; **#2** is the first column of the block; **#3** is the last row of the block; **#4** is the last column of the block; **#5** is a list of `key=value` options; **#6** is the label

```

6658 \cs_new_protected:Npn \@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6659   {

```

The group is for the keys.

```

6660   \group_begin:
6661   \int_compare:nNnT { #1 } = { #3 }
6662     { \str_set:Nn \l_@@_vpos_of_block_str { t } }
6663   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
6664   \bool_if:NT \l_@@_vlines_block_bool
6665     {
6666       \tl_gput_right:Nx \g_nicematrix_code_after_tl
6667         {
6668           \@_vlines_block:nnn
6669           { \exp_not:n { #5 } }
6670           { #1 - #2 }
6671           { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6672         }
6673     }
6674   \bool_if:NT \l_@@_hlines_block_bool
6675     {
6676       \tl_gput_right:Nx \g_nicematrix_code_after_tl
6677         {
6678           \@_hlines_block:nnn
6679           { \exp_not:n { #5 } }
6680           { #1 - #2 }
6681           { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6682         }
6683     }
6684   \bool_if:nF
6685     {
6686       \l_@@_transparent_bool

```

```

6687     || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
6688   }
6689   {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

6690     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6691     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6692   }

```

```

6693 \bool_lazy_and:nnT
6694 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
6695 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
6696 { \@@_error:n { hlines-with-color } }

```

```

6697 \tl_if_empty:NF \l_@@_draw_tl
6698 {
6699   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6700   {
6701     \@@_stroke_block:nnn
6702     { \exp_not:n { #5 } }
6703     { #1 - #2 }
6704     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6705   }
6706   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6707   { { #1 } { #2 } { #3 } { #4 } }
6708 }

```

```

6709 \clist_if_empty:NF \l_@@_borders_clist
6710 {
6711   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6712   {
6713     \@@_stroke_borders_block:nnn
6714     { \exp_not:n { #5 } }
6715     { #1 - #2 }
6716     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6717   }
6718 }

```

```

6719 \tl_if_empty:NF \l_@@_fill_tl
6720 {
6721   \tl_gput_right:Nx \g_@@_pre_code_before_tl
6722   {
6723     \exp_not:N \roundedrectanglecolor
6724     \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
6725       { \l_@@_fill_tl }
6726       { { \l_@@_fill_tl } }
6727       { #1 - #2 }
6728       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6729       { \dim_use:N \l_@@_rounded_corners_dim }
6730     ]
6731   }

```

```

6732 \seq_if_empty:NF \l_@@_tikz_seq
6733 {
6734   \tl_gput_right:Nx \g_nicematrix_code_before_tl
6735   {
6736     \@@_block_tikz:nnnnn
6737     { #1 }
6738     { #2 }
6739     { \int_use:N \l_@@_last_row_int }
6740     { \int_use:N \l_@@_last_col_int }
6741     { \seq_use:Nn \l_@@_tikz_seq { , } }
6742   }
6743 }

```

```

6744 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6745 {
6746   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6747   {
6748     \@@_actually_diagbox:nnnnnn
6749     { #1 }
6750     { #2 }
6751     { \int_use:N \l_@@_last_row_int }
6752     { \int_use:N \l_@@_last_col_int }
6753     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6754   }
6755 }

6756 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6757 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & & two & \\\
three & & four & five & \\\
six & & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node `1-1-block`

|           |       |
|-----------|-------|
| our block | one   |
| three     | two   |
| four      | five  |
| six       | seven |
|           | eight |

We highlight the node `1-1-block-short`

|           |       |
|-----------|-------|
| our block | one   |
| three     | two   |
| four      | five  |
| six       | seven |
|           | eight |

The construction of the node corresponding to the merged cells.

```

6758 \pgfpicture
6759 \pgfrememberpicturepositiononpagetrue
6760 \pgf@relevantforpicturesizefalse
6761 \@@_qpoint:n { row - #1 }
6762 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6763 \@@_qpoint:n { col - #2 }
6764 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6765 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6766 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6767 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6768 \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.

```

6769 \@@_pgf_rect_node:nnnnn
6770 { \@@_env: - #1 - #2 - block }
6771 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6772 \str_if_empty:NF \l_@@_block_name_str
6773 {
6774   \pgfnodealias
6775   { \@@_env: - \l_@@_block_name_str }
6776   { \@@_env: - #1 - #2 - block }
6777   \str_if_empty:NF \l_@@_name_str
6778   {
6779     \pgfnodealias

```



```

6780         { \l_@@_name_str - \l_@@_block_name_str }
6781         { \@@_env: - #1 - #2 - block }
6782     }
6783 }

```

Now, we create the “short node” which, in general, will be used to put the label (that is to say the content of the node). However, if one the keys L, C or R is used (that information is provided by the boolean `\l_@@_hpos_of_block_cap_bool`), we don’t need to create that node since the normal node is used to put the label.

```

6784     \bool_if:NF \l_@@_hpos_of_block_cap_bool
6785     {
6786         \dim_set_eq:NN \l_tmpb_dim \c_max_dim

```

The short node is constructed by taking into account the *contents* of the columns involved in at least one cell of the block. That’s why we have to do a loop over the rows of the array.

```

6787         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6788         {

```

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.

```

6789             \cs_if_exist:cT
6790             { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6791             {
6792                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6793                 {
6794                     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
6795                     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
6796                 }
6797             }
6798         }

```

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.

```

6799         \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
6800         {
6801             \@@_qpoint:n { col - #2 }
6802             \dim_set_eq:NN \l_tmpb_dim \pgf@x
6803         }
6804         \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
6805         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6806         {
6807             \cs_if_exist:cT
6808             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6809             {
6810                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6811                 {
6812                     \pgfpointanchor
6813                     { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6814                     { east }
6815                     \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
6816                 }
6817             }
6818         }
6819         \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
6820         {
6821             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6822             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6823         }
6824         \@@_pgf_rect_node:nnnnn
6825         { \@@_env: - #1 - #2 - block - short }
6826         \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6827     }

```

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.

```

6828 \bool_if:N\T \l_@@_medium_nodes_bool
6829 {
6830   \@@_pgf_rect_node:nnn
6831   { \@@_env: - #1 - #2 - block - medium }
6832   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
6833   {
6834     \pgfpointanchor
6835     { \@@_env:
6836       - \int_use:N \l_@@_last_row_int
6837       - \int_use:N \l_@@_last_col_int - medium
6838     }
6839     { south-east }
6840   }
6841 }

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

6842 \int_compare:nNnT { #1 } = 0
6843 {
6844   \int_compare:nNnT { #3 } = 0
6845   { \l_@@_code_for_first_row_tl }
6846 }
6847 \int_compare:nNnT { #1 } = \l_@@_last_row_int
6848 {
6849   \int_compare:nNnT { #3 } = \l_@@_last_row_int
6850   { \l_@@_code_for_last_row_tl }
6851 }

```

Now, we will put the label of the block.

```

6852 \bool_lazy_any:nTF
6853 {
6854   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
6855   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
6856   { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
6857 }
6858 % \medskip
6859 % \begin{macrocode}
6860 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

6861 \int_compare:nNnT { #2 } = 0
6862 { \str_set:Nn \l_@@_hpos_block_str r }
6863 \bool_if:nT \g_@@_last_col_found_bool
6864 {
6865   \int_compare:nNnT { #2 } = \g_@@_col_total_int
6866   { \str_set:Nn \l_@@_hpos_block_str l }
6867 }
6868 \tl_set:Nx \l_tmpa_tl
6869 {
6870   \str_case:Vn \l_@@_vpos_of_block_str
6871   {
6872     c {
6873       \str_case:Vn \l_@@_hpos_block_str
6874       {
6875         c { center }
6876         l { west }
6877         r { east }
6878       }
6879     }
6880   }
6881   T {
6882     \str_case:Vn \l_@@_hpos_block_str

```

```

6883         {
6884             c { north }
6885             l { north-west }
6886             r { north-east }
6887         }
6888     }
6889     B {
6890         \str_case:Vn \l_@@_hpos_block_str
6891         {
6892             c { south}
6893             l { south-west }
6894             r { south-east }
6895         }
6896     }
6897 }
6898 }
6899 }
6900 }
6901 \pgftransformshift
6902 {
6903     \pgfpointanchor
6904     {
6905         \@@_env: - #1 - #2 - block
6906         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6907     }
6908     { \l_tmpa_tl }
6909 }
6910 \pgfset { inner-xsep = \c_zero_dim }
6911 \pgfnode
6912 { rectangle }
6913 { \l_tmpa_tl }
6914 { \box_use_drop:N \l_@@_cell_box } { } { }
6915 }
6916 {
6917     \pgfextracty \l_tmpa_dim
6918     {
6919         \@@_qpoint:n
6920         {
6921             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
6922             - base
6923         }
6924     }
6925     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth } % added 2023-02-21

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

6926     \pgfpointanchor
6927     {
6928         \@@_env: - #1 - #2 - block
6929         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6930     }
6931     {
6932         \str_case:Vn \l_@@_hpos_block_str
6933         {
6934             c { center }
6935             l { west }
6936             r { east }
6937         }
6938     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

6939     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
6940     \pgfset { inner-sep = \c_zero_dim }
6941     \pgfnode

```

```

6942     { rectangle }
6943     {
6944         \str_case:Vn \l_@@_hpos_block_str
6945         {
6946             c { base }
6947             l { base~west }
6948             r { base~east }
6949         }
6950     }
6951     { \box_use_drop:N \l_@@_cell_box } { } { }
6952 }
6953 \endpgfpicture
6954 \group_end:
6955 }

```

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).

```

6956 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
6957 {
6958     \group_begin:
6959     \tl_clear:N \l_@@_draw_tl
6960     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6961     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
6962     \pgfpicture
6963     \pgfrememberpicturepositiononpagetrue
6964     \pgf@relevantforpicturesizefalse
6965     \tl_if_empty:NF \l_@@_draw_tl
6966     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

6967     \str_if_eq:VnTF \l_@@_draw_tl { default }
6968     { \CT@arc@ }
6969     { \@@_color:V \l_@@_draw_tl }
6970 }
6971 \pgfsetcornersarced
6972 {
6973     \pgfpoint
6974     { \dim_use:N \l_@@_rounded_corners_dim }
6975     { \dim_use:N \l_@@_rounded_corners_dim }
6976 }
6977 \@@_cut_on_hyphen:w #2 \q_stop
6978 \bool_lazy_and:nnT
6979 { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
6980 { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
6981 {
6982     \@@_qpoint:n { row - \l_tmpa_tl }
6983     \dim_set:Nn \l_tmpb_dim { \pgf@y }
6984     \@@_qpoint:n { col - \l_tmpb_tl }
6985     \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
6986     \@@_cut_on_hyphen:w #3 \q_stop
6987     \int_compare:nNnT \l_tmpa_tl > \c@iRow
6988     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
6989     \int_compare:nNnT \l_tmpb_tl > \c@jCol
6990     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
6991     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
6992     \dim_set:Nn \l_tmpa_dim { \pgf@y }
6993     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6994     \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
6995     \pgfpathrectanglecorners
6996     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6997     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }

```

```

6998     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
6999     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7000       { \pgfusepathqstroke }
7001       { \pgfusepath { stroke } }
7002   }
7003 \endpgfpicture
7004 \group_end:
7005 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7006 \keys_define:nn { NiceMatrix / BlockStroke }
7007 {
7008   color .tl_set:N = \l_@@_draw_tl ,
7009   draw .tl_set:N = \l_@@_draw_tl ,
7010   draw .default:n = default ,
7011   line-width .dim_set:N = \l_@@_line_width_dim ,
7012   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7013   rounded-corners .default:n = 4 pt
7014 }

```

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

7015 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7016 {
7017   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7018   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7019   \@@_cut_on_hyphen:w #2 \q_stop
7020   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7021   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7022   \@@_cut_on_hyphen:w #3 \q_stop
7023   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7024   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7025   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7026     {
7027       \use:x
7028       {
7029         \@@_vline:n
7030         {
7031           position = ##1 ,
7032           start = \l_@@_tmpc_tl ,
7033           end = \int_eval:n { \l_tmpa_tl - 1 } ,
7034           total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7035         }
7036       }
7037     }
7038 }
7039 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7040 {
7041   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7042   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7043   \@@_cut_on_hyphen:w #2 \q_stop
7044   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7045   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7046   \@@_cut_on_hyphen:w #3 \q_stop
7047   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7048   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7049   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7050     {
7051       \use:x
7052       {
7053         \@@_hline:n
7054         {

```

```

7055         position = ##1 ,
7056         start = \l_@@_tmpd_tl ,
7057         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7058         total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7059     }
7060 }
7061 }
7062 }

```

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).

```

7063 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7064 {
7065     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7066     \keys_set:known:n { NiceMatrix / BlockBorders } { #1 }
7067     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7068     { \@@_error:n { borders~forbidden } }
7069     {
7070         \tl_clear_new:N \l_@@_borders_tikz_tl
7071         \keys_set:nV
7072             { NiceMatrix / OnlyForTikzInBorders }
7073             \l_@@_borders_clist
7074         \@@_cut_on_hyphen:w #2 \q_stop
7075         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7076         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7077         \@@_cut_on_hyphen:w #3 \q_stop
7078         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7079         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7080         \@@_stroke_borders_block_i:
7081     }
7082 }
7083 \hook_gput_code:nnn { begindocument } { . }
7084 {
7085     \cs_new_protected:Npx \@@_stroke_borders_block_i:
7086     {
7087         \c_@@_pgfortikzpicture_tl
7088         \@@_stroke_borders_block_ii:
7089         \c_@@_endpgfortikzpicture_tl
7090     }
7091 }
7092 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7093 {
7094     \pgfrememberpicturepositiononpagetrue
7095     \pgf@relevantforpicturesizefalse
7096     \CT@arc@
7097     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7098     \clist_if_in:NnT \l_@@_borders_clist { right }
7099     { \@@_stroke_vertical:n \l_tmpb_tl }
7100     \clist_if_in:NnT \l_@@_borders_clist { left }
7101     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7102     \clist_if_in:NnT \l_@@_borders_clist { bottom }
7103     { \@@_stroke_horizontal:n \l_tmpa_tl }
7104     \clist_if_in:NnT \l_@@_borders_clist { top }
7105     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7106 }
7107 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7108 {
7109     tikz .code:n =
7110     \cs_if_exist:NTF \tikzpicture
7111     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7112     { \@@_error:n { tikz-in~borders~without~tikz } } ,

```

```

7113 tikz .value_required:n = true ,
7114 top .code:n = ,
7115 bottom .code:n = ,
7116 left .code:n = ,
7117 right .code:n = ,
7118 unknown .code:n = \@@_error:n { bad-border }
7119 }

```

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).

```

7120 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7121 {
7122   \@@_qpoint:n \l_@@_tmpc_tl
7123   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7124   \@@_qpoint:n \l_tmpa_tl
7125   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7126   \@@_qpoint:n { #1 }
7127   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7128   {
7129     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7130     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7131     \pgfusepathqstroke
7132   }
7133   {
7134     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7135     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7136   }
7137 }

```

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).

```

7138 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7139 {
7140   \@@_qpoint:n \l_@@_tmpd_tl
7141   \clist_if_in:NnTF \l_@@_borders_clist { left }
7142   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7143   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7144   \@@_qpoint:n \l_tmpb_tl
7145   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7146   \@@_qpoint:n { #1 }
7147   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7148   {
7149     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7150     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7151     \pgfusepathqstroke
7152   }
7153   {
7154     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7155     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7156   }
7157 }

```

Here is the set of keys for the command \@@\_stroke\_borders\_block:nnn.

```

7158 \keys_define:nn { NiceMatrix / BlockBorders }
7159 {
7160   borders .clist_set:N = \l_@@_borders_clist ,
7161   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7162   rounded-corners .default:n = 4 pt ,
7163   line-width .dim_set:N = \l_@@_line_width_dim ,
7164 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`. The arguments `#1` and `#2` are the coordinates of the first cell and `#3` and `#4` the coordinates of the last cell of the block. `#5` is a comma-separated list of the Tikz keys used with the path.

```

7165 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7166   {
7167     \begin { tikzpicture }
7168     \clist_map_inline:nn { #5 }
7169       {
7170         \path [ ##1 ]
7171           ( #1 -| #2 )
7172           rectangle
7173           ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
7174       }
7175     \end { tikzpicture }
7176   }

```

## How to draw the dotted lines transparently

```

7177 \cs_set_protected:Npn \@@_renew_matrix:
7178   {
7179     \RenewDocumentEnvironment { pmatrix } { }
7180       { \pNiceMatrix }
7181       { \endpNiceMatrix }
7182     \RenewDocumentEnvironment { vmatrix } { }
7183       { \vNiceMatrix }
7184       { \endvNiceMatrix }
7185     \RenewDocumentEnvironment { Vmatrix } { }
7186       { \VNiceMatrix }
7187       { \endVNiceMatrix }
7188     \RenewDocumentEnvironment { bmatrix } { }
7189       { \bNiceMatrix }
7190       { \endbNiceMatrix }
7191     \RenewDocumentEnvironment { Bmatrix } { }
7192       { \BNiceMatrix }
7193       { \endBNiceMatrix }
7194   }

```

## Automatic arrays

We will extract the potential keys `columns-type`, `l`, `c`, `r` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7195 \keys_define:nn { NiceMatrix / Auto }
7196   {
7197     columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
7198     columns-type .value_required:n = true ,
7199     l .meta:n = { columns-type = l } ,
7200     r .meta:n = { columns-type = r } ,
7201     c .meta:n = { columns-type = c } ,
7202     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7203     delimiters / color .value_required:n = true ,
7204     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7205     delimiters / max-width .default:n = true ,
7206     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7207     delimiters .value_required:n = true ,
7208   }
7209 \NewDocumentCommand \AutoNiceMatrixWithDelims
7210   { m m 0 { } > { \SplitArgument { 1 } { - } m 0 { } m ! 0 { } }
7211   { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7212 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7213   {

```



The group is for the protection of the keys.

```

7214 \group_begin:
7215 \bool_set_true:N \l_@@_Matrix_bool
7216 \keys_set_known:nN { NiceMatrix / Auto } { #6 } \l_tmpa_tl

```

We nullify the command `\@@_transform_preamble:` because we will provide a preamble which is yet transformed (by using `\l_@@_columns_type_tl` which is yet nicematrix-ready).

```

7217 \cs_set_eq:NN \@@_transform_preamble: \prg_do_nothing:
7218 \use:x
7219 {
7220   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7221   { * { #4 } { \exp_not:V \l_@@_columns_type_tl } }
7222   [ \exp_not:V \l_tmpa_tl ]
7223 }
7224 \int_compare:nNnT \l_@@_first_row_int = 0
7225 {
7226   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7227   \prg_replicate:nn { #4 - 1 } { & }
7228   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7229 }
7230 \prg_replicate:nn { #3 }
7231 {
7232   \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

We put `{ }` before `#6` to avoid a hasty expansion of a potential `\arabic{iRow}` at the beginning of the row which would result in an incorrect value of that `iRow` (since `iRow` is incremented in the first cell of the row of the `\halign`).

```

7233   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7234   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7235 }
7236 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7237 {
7238   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7239   \prg_replicate:nn { #4 - 1 } { & }
7240   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7241 }
7242 \end { NiceArrayWithDelims }
7243 \group_end:
7244 }
7245 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7246 {
7247   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7248   {
7249     \bool_gset_false:N \g_@@_NiceArray_bool
7250     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7251     \AutoNiceMatrixWithDelims { #2 } { #3 }
7252   }
7253 }
7254 \@@_define_com:nnn p ( )
7255 \@@_define_com:nnn b [ ]
7256 \@@_define_com:nnn v | |
7257 \@@_define_com:nnn V \ | \ |
7258 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7259 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7260 {
7261   \group_begin:
7262   \bool_gset_true:N \g_@@_NiceArray_bool
7263   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7264   \group_end:
7265 }

```

## The redefinition of the command `\dotfill`

```
7266 \cs_set_eq:NN \@@_old_dotfill \dotfill
7267 \cs_new_protected:Npn \@@_dotfill:
7268 {
```

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}`).

```
7269   \@@_old_dotfill
7270   \bool_if:NT \l_@@_NiceTabular_bool
7271     { \group_insert_after:N \@@_dotfill_ii: }
7272     { \group_insert_after:N \@@_dotfill_i: }
7273   }
7274 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
7275 \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`.

```
7276 \cs_new_protected:Npn \@@_dotfill_iii:
7277 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }
```

## The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```
7278 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7279 {
7280   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7281   {
7282     \@@_actually_diagbox:nnnnnn
7283     { \int_use:N \c@iRow }
7284     { \int_use:N \c@jCol }
7285     { \int_use:N \c@iRow }
7286     { \int_use:N \c@jCol }
7287     { \exp_not:n { #1 } }
7288     { \exp_not:n { #2 } }
7289   }
```

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`.

```
7290   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7291   {
7292     { \int_use:N \c@iRow }
7293     { \int_use:N \c@jCol }
7294     { \int_use:N \c@iRow }
7295     { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
7296     { }
7297   }
7298 }
```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it’s possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```
7299 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7300 {
7301   \pgfpicture
7302   \pgf@relevantforpicturesizefalse
7303   \pgfrememberpicturepositiononpagetrue
7304   \@@_qpoint:n { row - #1 }
```

```

7305 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7306 \@@_qpoint:n { col - #2 }
7307 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7308 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7309 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7310 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7311 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7312 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7313 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7314 {

```

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.

```

7315 \CT@arc@
7316 \pgfsetroundcap
7317 \pgfusepathqstroke
7318 }
7319 \pgfset { inner~sep = 1 pt }
7320 \pgfscope
7321 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7322 \pgfnode { rectangle } { south-west }
7323 {
7324 \begin { minipage } { 20 cm }
7325 \@@_math_toggle_token: #5 \@@_math_toggle_token:
7326 \end { minipage }
7327 }
7328 { }
7329 { }
7330 \endpgfscope
7331 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7332 \pgfnode { rectangle } { north-east }
7333 {
7334 \begin { minipage } { 20 cm }
7335 \raggedleft
7336 \@@_math_toggle_token: #6 \@@_math_toggle_token:
7337 \end { minipage }
7338 }
7339 { }
7340 { }
7341 \endpgfpicture
7342 }

```

## 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.

```

7343 \keys_define:nn { NiceMatrix }
7344 {
7345 CodeAfter / rules .inherit:n = NiceMatrix / rules ,
7346 CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
7347 }
7348 \keys_define:nn { NiceMatrix / CodeAfter }
7349 {
7350 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
7351 sub-matrix .value_required:n = true ,
7352 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7353 delimiters / color .value_required:n = true ,
7354 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7355 rules .value_required:n = true ,
7356 unknown .code:n = \@@_error:n { Unknown-key-for~CodeAfter }
7357 }

```

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. 136.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
7358 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_ii:n }
```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_ii:n` which begins with `\`.

```
7359 \cs_new_protected:Npn \@@_CodeAfter_i: { \ \omit \@@_CodeAfter_ii:n }
```

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```
7360 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7361 {
7362   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7363   \@@_CodeAfter_iv:n
7364 }
```

We catch the argument of the command `\end` (in `#1`).

```
7365 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7366 {
```

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.

```
7367   \str_if_eq:eeTF \@currenvir { #1 }
7368     { \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`.

```
7369   {
7370     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7371     \@@_CodeAfter_ii:n
7372   }
7373 }
```

## The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
7374 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7375 {
7376   \pgfpicture
7377   \pgfrememberpicturepositiononpagetrue
7378   \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.

```
7379   \@@_qpoint:n { row - 1 }
7380   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7381   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7382   \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).

```

7383 \bool_if:nTF { #3 }
7384   { \dim_set:eq:NN \l_tmpa_dim \c_max_dim }
7385   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7386 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7387   {
7388     \cs_if_exist:cT
7389     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7390     {
7391       \pgfpointanchor
7392       { \@@_env: - ##1 - #2 }
7393       { \bool_if:nTF { #3 } { west } { east } }
7394       \dim_set:Nn \l_tmpa_dim
7395       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7396     }
7397   }

```

Now we can put the delimiter with a node of PGF.

```

7398 \pgfset { inner~sep = \c_zero_dim }
7399 \dim_zero:N \nulldelimiterspace
7400 \pgftransformshift
7401   {
7402     \pgfpoint
7403     { \l_tmpa_dim }
7404     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7405   }
7406 \pgfnode
7407   { rectangle }
7408   { \bool_if:nTF { #3 } { east } { west } }
7409   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

7410 \nullfont
7411 \c_math_toggle_token
7412 \@@_color:V \l_@@_delimiters_color_tl
7413 \bool_if:nTF { #3 } { \left #1 } { \left . }
7414 \vcenter
7415   {
7416     \nullfont
7417     \hrule \@height
7418       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7419       \@depth \c_zero_dim
7420       \@width \c_zero_dim
7421   }
7422 \bool_if:nTF { #3 } { \right . } { \right #1 }
7423 \c_math_toggle_token
7424 }
7425 { }
7426 { }
7427 \endpgfpicture
7428 }

```

## The command `\SubMatrix`

```

7429 \keys_define:nn { NiceMatrix / sub-matrix }
7430   {
7431     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7432     extra-height .value_required:n = true ,
7433     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7434     left-xshift .value_required:n = true ,
7435     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7436     right-xshift .value_required:n = true ,

```

```

7437 xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7438 xshift .value_required:n = true ,
7439 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7440 delimiters / color .value_required:n = true ,
7441 slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7442 slim .default:n = true ,
7443 hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7444 hlines .default:n = all ,
7445 vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7446 vlines .default:n = all ,
7447 hvlines .meta:n = { hlines, vlines } ,
7448 hvlines .value_forbidden:n = true ,
7449 }
7450 \keys_define:nn { NiceMatrix }
7451 {
7452   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7453   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7454   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7455   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7456   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7457   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7458 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

7459 \keys_define:nn { NiceMatrix / SubMatrix }
7460 {
7461   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7462   delimiters / color .value_required:n = true ,
7463   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7464   hlines .default:n = all ,
7465   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7466   vlines .default:n = all ,
7467   hvlines .meta:n = { hlines, vlines } ,
7468   hvlines .value_forbidden:n = true ,
7469   name .code:n =
7470     \tl_if_empty:nTF { #1 }
7471     { \@@_error:n { Invalid-name } }
7472     {
7473       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7474       {
7475         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7476         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7477         {
7478           \str_set:Nn \l_@@_submatrix_name_str { #1 }
7479           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7480         }
7481       }
7482       { \@@_error:n { Invalid-name } }
7483     } ,
7484   name .value_required:n = true ,
7485   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7486   rules .value_required:n = true ,
7487   code .tl_set:N = \l_@@_code_tl ,
7488   code .value_required:n = true ,
7489   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
7490 }

7491 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
7492 {
7493   \peek_remove_spaces:n
7494   {
7495     \tl_gput_right:Nx \g_@@_pre_code_after_tl

```

```

7496     {
7497         \SubMatrix { #1 } { #2 } { #3 } { #4 }
7498         [
7499             delimiters / color = \l_@@_delimiters_color_tl ,
7500             hlines = \l_@@_submatrix_hlines_clist ,
7501             vlines = \l_@@_submatrix_vlines_clist ,
7502             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
7503             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
7504             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
7505             slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
7506             #5
7507         ]
7508     }
7509     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
7510 }
7511 }
7512 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
7513 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7514 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
7515 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
7516 {
7517     \seq_gput_right:Nx \g_@@_submatrix_seq
7518     {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

7519     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
7520     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
7521     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
7522     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
7523 }
7524 }

```

In the pre-code-after and in the `\CodeAfter` the following command `\@@_SubMatrix` will be linked to `\SubMatrix`.

- #1 is the left delimiter;
- #2 is the upper-left cell of the matrix with the format  $i-j$ ;
- #3 is the lower-right cell of the matrix with the format  $i-j$ ;
- #4 is the right delimiter;
- #5 is the list of options of the command;
- #6 is the potential subscript;
- #7 is the potential superscript.

For explanations about the construction with rescanning of the preamble, see the documentation for the user command `\Cdots`.

```

7525 \hook_gput_code:nnn { begindocument } { . }
7526 {
7527     \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
7528     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
7529     \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
7530     {
7531         \peek_remove_spaces:n
7532         {
7533             \@@_sub_matrix:nnnnnnn
7534             { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
7535         }
7536     }
7537 }

```

The following macro will compute  $\backslash l_{@@\_first\_i\_tl}$ ,  $\backslash l_{@@\_first\_j\_tl}$ ,  $\backslash l_{@@\_last\_i\_tl}$  and  $\backslash l_{@@\_last\_j\_tl}$  from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

7538 \NewDocumentCommand \@@_compute_i_j:nn
7539   { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7540   { \@@_compute_i_j:nnnn #1 #2 }

7541 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
7542   {
7543     \tl_set:Nn \l_@@_first_i_tl { #1 }
7544     \tl_set:Nn \l_@@_first_j_tl { #2 }
7545     \tl_set:Nn \l_@@_last_i_tl { #3 }
7546     \tl_set:Nn \l_@@_last_j_tl { #4 }
7547     \tl_if_eq:NnT \l_@@_first_i_tl { last }
7548       { \tl_set:NV \l_@@_first_i_tl \c@iRow }
7549     \tl_if_eq:NnT \l_@@_first_j_tl { last }
7550       { \tl_set:NV \l_@@_first_j_tl \c@jCol }
7551     \tl_if_eq:NnT \l_@@_last_i_tl { last }
7552       { \tl_set:NV \l_@@_last_i_tl \c@iRow }
7553     \tl_if_eq:NnT \l_@@_last_j_tl { last }
7554       { \tl_set:NV \l_@@_last_j_tl \c@jCol }
7555   }

7556 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
7557   {
7558     \group_begin:

```

The four following token lists correspond to the position of the  $\backslash$ SubMatrix.

```

7559   \@@_compute_i_j:nn { #2 } { #3 }
7560   \bool_lazy_or:nnTF
7561     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7562     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7563     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
7564     {
7565       \str_clear_new:N \l_@@_submatrix_name_str
7566       \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
7567       \pgfpicture
7568       \pgfrememberpicturerepositiononpagetrue
7569       \pgf@relevantforpicturesizefalse
7570       \pgfset { inner-sep = \c_zero_dim }
7571       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7572       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of  $\backslash$ int\_step\_inline:nnn is provided by currfication.

```

7573   \bool_if:NTF \l_@@_submatrix_slim_bool
7574     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
7575     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
7576     {
7577       \cs_if_exist:cT
7578         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7579         {
7580           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7581           \dim_set:Nn \l_@@_x_initial_dim
7582             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7583         }
7584       \cs_if_exist:cT
7585         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7586         {
7587           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7588           \dim_set:Nn \l_@@_x_final_dim
7589             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7590         }
7591     }
7592   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
7593     { \@@_error:nn { Impossible-delimiter } { left } }

```



```

7594     {
7595         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
7596         { \@@_error:nn { Impossible~delimiter } { right } }
7597         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
7598     }
7599     \endpgfpicture
7600 }
7601 \group_end:
7602 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

7603 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
7604 {
7605     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
7606     \dim_set:Nn \l_@@_y_initial_dim
7607     {
7608         \fp_to_dim:n
7609         {
7610             \pgf@y
7611             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
7612         }
7613     } % modified 6.13c
7614     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
7615     \dim_set:Nn \l_@@_y_final_dim
7616     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
7617     % modified 6.13c
7618     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
7619     {
7620         \cs_if_exist:cT
7621         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
7622         {
7623             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
7624             \dim_set:Nn \l_@@_y_initial_dim
7625             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
7626         }
7627         \cs_if_exist:cT
7628         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
7629         {
7630             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
7631             \dim_set:Nn \l_@@_y_final_dim
7632             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
7633         }
7634     }
7635     \dim_set:Nn \l_tmpa_dim
7636     {
7637         \l_@@_y_initial_dim - \l_@@_y_final_dim +
7638         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
7639     }
7640     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

7641     \group_begin:
7642     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7643     \@@_set_CT@arc@:V \l_@@_rules_color_tl
7644     \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`.

```

7645     \seq_map_inline:Nn \g_@@_cols_vlism_seq
7646     {
7647         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
7648         {

```

```

7649     \int_compare:nNnT
7650     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
7651     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

7652         \@@_qpoint:n { col - ##1 }
7653         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7654         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7655         \pgfusepathqstroke
7656     }
7657 }
7658 }

```

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.

```

7659     \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
7660     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
7661     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
7662     {
7663     \bool_lazy_and:nnTF
7664     { \int_compare_p:nNn { ##1 } > 0 }
7665     {
7666         \int_compare_p:nNn
7667         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7668     {
7669         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7670         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7671         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7672         \pgfusepathqstroke
7673     }
7674     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
7675 }

```

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.

```

7676     \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7677     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7678     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
7679     {
7680     \bool_lazy_and:nnTF
7681     { \int_compare_p:nNn { ##1 } > 0 }
7682     {
7683         \int_compare_p:nNn
7684         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7685     {
7686         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7687     \group_begin:

```

We compute in `\l_tmpa_dim` the  $x$ -value of the left end of the rule.

```

7688     \dim_set:Nn \l_tmpa_dim
7689     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7690     \str_case:nn { #1 }
7691     {
7692         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7693         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7694         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7695     }
7696     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

7697     \dim_set:Nn \l_tmpb_dim
7698     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7699     \str_case:nn { #2 }

```

```

7700         {
7701         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7702         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7703         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7704         }
7705         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7706         \pgfusepathqstroke
7707         \group_end:
7708     }
7709     { \@@_error:nmn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7710 }

```

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).

```

7711     \str_if_empty:NF \l_@@_submatrix_name_str
7712     {
7713         \@@_pgf_rect_node:nnnn \l_@@_submatrix_name_str
7714         \l_@@_x_initial_dim \l_@@_y_initial_dim
7715         \l_@@_x_final_dim \l_@@_y_final_dim
7716     }
7717     \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`.

```

7718     \begin { pgfscope }
7719     \pgftransformshift
7720     {
7721         \pgfpoint
7722         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7723         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7724     }
7725     \str_if_empty:NTF \l_@@_submatrix_name_str
7726     { \@@_node_left:nn #1 { } }
7727     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7728     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7729     \pgftransformshift
7730     {
7731         \pgfpoint
7732         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7733         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7734     }
7735     \str_if_empty:NTF \l_@@_submatrix_name_str
7736     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7737     {
7738         \@@_node_right:nnnn #2
7739         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7740     }
7741     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7742     \flag_clear_new:n { nicematrix }
7743     \l_@@_code_tl
7744 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the  $i$  and  $j$  in specifications of nodes of the forms  $i-j$ ,  $row-i$ ,  $col-j$  and  $i-|j$  refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

7745 \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.

```

7746 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
7747   {
7748     \use:e
7749     { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
7750   }

```

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 -.

```

7751 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
7752   { #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.

```

7753 \tl_const:Nn \c_@@_integers_alist_tl
7754   {
7755     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7756     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7757     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7758     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7759   }

```

```

7760 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
7761   {

```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form  $i-j$ . In that case, the  $i$  of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the  $j$  arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

7762   \tl_if_empty:nTF { #2 }
7763     {
7764       \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7765         {
7766           \flag_raise:n { nicematrix }
7767           \int_if_even:nTF { \flag_height:n { nicematrix } }
7768             { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7769             { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7770         }
7771       { #1 }
7772     }

```

If there is an hyphen, we have to see whether we have a node of the form  $i-j$ ,  $row-i$  or  $col-j$ .

```

7773   { \@@_pgfpointanchor_iii:w { #1 } #2 }
7774   }

```

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`).

```

7775 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
7776   {
7777     \str_case:nnF { #1 }
7778       {
7779         { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7780         { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7781       }

```

Now the case of a node of the form  $i-j$ .

```

7782     {
7783     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7784     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7785     }
7786   }

```

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`).

```

7787 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7788   {
7789   \pgfnode
7790     { rectangle }
7791     { east }
7792     {
7793     \nullfont
7794     \c_math_toggle_token
7795     \@@_color:V \l_@@_delimiters_color_tl
7796     \left #1
7797     \vcenter
7798     {
7799     \nullfont
7800     \hrule \@height \l_tmpa_dim
7801     \@depth \c_zero_dim
7802     \@width \c_zero_dim
7803     }
7804     \right .
7805     \c_math_toggle_token
7806     }
7807     { #2 }
7808     { }
7809   }

```

The command `\@@_node_right:nnn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

7810 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
7811   {
7812   \pgfnode
7813     { rectangle }
7814     { west }
7815     {
7816     \nullfont
7817     \c_math_toggle_token
7818     \@@_color:V \l_@@_delimiters_color_tl
7819     \left .
7820     \vcenter
7821     {
7822     \nullfont
7823     \hrule \@height \l_tmpa_dim
7824     \@depth \c_zero_dim
7825     \@width \c_zero_dim
7826     }
7827     \right #1
7828     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
7829     ^ { \smash { #4 } }
7830     \c_math_toggle_token
7831     }
7832     { #2 }
7833     { }
7834   }

```

## Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

7835 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
7836 {
7837   \peek_remove_spaces:n
7838   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
7839 }

7840 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
7841 {
7842   \peek_remove_spaces:n
7843   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
7844 }

7845 \keys_define:nn { NiceMatrix / Brace }
7846 {
7847   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
7848   left-shorten .default:n = true ,
7849   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
7850   shorten .meta:n = { left-shorten , right-shorten } ,
7851   right-shorten .default:n = true ,
7852   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
7853   yshift .value_required:n = true ,
7854   yshift .initial:n = \c_zero_dim ,
7855   color .tl_set:N = \l_tmpa_tl ,
7856   color .value_required:n = true ,
7857   unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
7858 }

```

`#1` is the first cell of the rectangle (with the syntax  $i-j$ ; `#2` is the last cell of the rectangle; `#3` is the label of the text; `#4` is the optional argument (a list of *key-value* pairs); `#5` is equal to `under` or `over`.

```

7859 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
7860 {
7861   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

7862   \@@_compute_i_j:nn { #1 } { #2 }
7863   \bool_lazy_or:nnTF
7864     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7865     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7866     {
7867       \str_if_eq:nnTF { #5 } { under }
7868       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
7869       { \@@_error:nn { Construct-too-large } { \OverBrace } }
7870     }
7871   {
7872     \tl_clear:N \l_tmpa_tl
7873     \keys_set:nn { NiceMatrix / Brace } { #4 }
7874     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
7875     \pgfpicture
7876     \pgfrememberpicturerepositiononpagetrue
7877     \pgf@relevantforpicturesizefalse
7878     \bool_if:NT \l_@@_brace_left_shorten_bool
7879     {
7880       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7881       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7882         {
7883           \cs_if_exist:cT
7884             { pgf @ sh @ ns @ \@@_env: - #1 - \l_@@_first_j_tl }
7885             {
7886               \pgfpointanchor { \@@_env: - #1 - \l_@@_first_j_tl } { west }

```

```

7887         \dim_set:Nn \l_@@_x_initial_dim
7888         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7889     }
7890 }
7891 }
7892 \bool_lazy_or:nnT
7893 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
7894 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
7895 {
7896     \@@_qpoint:n { col - \l_@@_first_j_tl }
7897     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7898 }
7899 \bool_if:NT \l_@@_brace_right_shorten_bool
7900 {
7901     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
7902     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7903     {
7904         \cs_if_exist:cT
7905         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7906         {
7907             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7908             \dim_set:Nn \l_@@_x_final_dim
7909             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7910         }
7911     }
7912 }
7913 \bool_lazy_or:nnT
7914 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
7915 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
7916 {
7917     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
7918     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7919 }
7920 \pgfset { inner-sep = \c_zero_dim }
7921 \str_if_eq:nnTF { #5 } { under }
7922 { \@@_underbrace_i:n { #3 } }
7923 { \@@_overbrace_i:n { #3 } }
7924 \endpgfpicture
7925 }
7926 \group_end:
7927 }

```

The argument is the text to put above the brace.

```

7928 \cs_new_protected:Npn \@@_overbrace_i:n #1
7929 {
7930     \@@_qpoint:n { row - \l_@@_first_i_tl }
7931     \pgftransformshift
7932     {
7933         \pgfpoint
7934         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7935         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
7936     }
7937     \pgfnode
7938     { rectangle }
7939     { south }
7940     {
7941         \vbox_top:n
7942         {
7943             \group_begin:
7944             \everycr { }
7945             \halign
7946             {
7947                 \hfil ## \hfil \crrc
7948                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr

```

```

7949         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
7950         \c_math_toggle_token
7951         \overbrace
7952         {
7953             \hbox_to_wd:nn
7954             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7955             { }
7956         }
7957         \c_math_toggle_token
7958         \cr
7959     }
7960 \group_end:
7961 }
7962 }
7963 { }
7964 { }
7965 }

```

The argument is the text to put under the brace.

```

7966 \cs_new_protected:Npn \@@_underbrace_i:n #1
7967 {
7968     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
7969     \pgftransformshift
7970     {
7971         \pgfpoint
7972         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7973         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
7974     }
7975     \pgfnode
7976     { rectangle }
7977     { north }
7978     {
7979         \group_begin:
7980         \everycr { }
7981         \vbox:n
7982         {
7983             \halign
7984             {
7985                 \hfil ## \hfil \crcr
7986                 \c_math_toggle_token
7987                 \underbrace
7988                 {
7989                     \hbox_to_wd:nn
7990                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7991                     { }
7992                 }
7993                 \c_math_toggle_token
7994                 \cr
7995                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
7996                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7997             }
7998         }
7999         \group_end:
8000     }
8001     { }
8002     { }
8003 }

```

## The command \ShowCellNames

```

8004 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }

```



```

8005 {
8006   \dim_zero_new:N \g_@@_tmpc_dim
8007   \dim_zero_new:N \g_@@_tmpd_dim
8008   \dim_zero_new:N \g_@@_tmpe_dim
8009   \int_step_inline:nn \c@iRow
8010   {
8011     \begin { pgfpicture }
8012     \@@_qpoint:n { row - ##1 }
8013     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8014     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8015     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8016     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8017     \bool_if:NTF \l_@@_in_code_after_bool
8018     \end { pgfpicture }
8019     \int_step_inline:nn \c@jCol
8020     {
8021       \hbox_set:Nn \l_tmpa_box
8022         { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8023       \begin { pgfpicture }
8024       \@@_qpoint:n { col - #####1 }
8025       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8026       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8027       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8028       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8029       \endpgfpicture
8030       \end { pgfpicture }
8031       \fp_set:Nn \l_tmpa_fp
8032       {
8033         \fp_min:nn
8034         {
8035           \fp_min:nn
8036           { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8037           { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8038         }
8039         { 1.0 }
8040       }
8041       \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8042       \pgfpicture
8043       \pgfrememberpicturepositiononpagetrue
8044       \pgf@relevantforpicturesizefalse
8045       \pgftransformshift
8046       {
8047         \pgfpoint
8048         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8049         { \dim_use:N \g_tmpa_dim }
8050       }
8051       \pgfnode
8052       { rectangle }
8053       { center }
8054       { \box_use:N \l_tmpa_box }
8055       { }
8056       { }
8057       \endpgfpicture
8058     }
8059   }
8060 }

8061 \NewDocumentCommand \@@_ShowCellNames { }
8062 {
8063   \bool_if:NT \l_@@_in_code_after_bool
8064   {
8065     \pgfpicture
8066     \pgfrememberpicturepositiononpagetrue
8067     \pgf@relevantforpicturesizefalse

```

```

8068 \pgfpathrectanglecorners
8069   { \@@_qpoint:n { 1 } }
8070   { \@@_qpoint:n { \int_eval:n { \c@iRow + 1 } } }
8071 \pgfsetfillopacity { 0.75 }
8072 \pgfsetfillcolor { white }
8073 \pgfusepathqfill
8074 \endpgfpicture
8075 }
8076 \dim_zero_new:N \g_@@_tmpc_dim
8077 \dim_zero_new:N \g_@@_tmpd_dim
8078 \dim_zero_new:N \g_@@_tmpe_dim
8079 \int_step_inline:nn \c@iRow
8080 {
8081   \bool_if:NTF \l_@@_in_code_after_bool
8082   {
8083     \pgfpicture
8084     \pgfrememberpicturepositiononpagetrue
8085     \pgf@relevantforpicturesizefalse
8086   }
8087   { \begin { pgfpicture } }
8088   \@@_qpoint:n { row - ##1 }
8089   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8090   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8091   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8092   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8093   \bool_if:NTF \l_@@_in_code_after_bool
8094   { \endpgfpicture }
8095   { \end { pgfpicture } }
8096   \int_step_inline:nn \c@jCol
8097   {
8098     \hbox_set:Nn \l_tmpa_box
8099     {
8100       \normalfont \Large \sffamily \bfseries
8101       \bool_if:NTF \l_@@_in_code_after_bool
8102       { \color { red } }
8103       { \color { red ! 50 } }
8104       ##1 - #####1
8105     }
8106     \bool_if:NTF \l_@@_in_code_after_bool
8107     {
8108       \pgfpicture
8109       \pgfrememberpicturepositiononpagetrue
8110       \pgf@relevantforpicturesizefalse
8111     }
8112     { \begin { pgfpicture } }
8113     \@@_qpoint:n { col - #####1 }
8114     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8115     \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8116     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8117     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8118     \bool_if:NTF \l_@@_in_code_after_bool
8119     { \endpgfpicture }
8120     { \end { pgfpicture } }
8121     \fp_set:Nn \l_tmpa_fp
8122     {
8123       \fp_min:nn
8124       {
8125         \fp_min:nn
8126         { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8127         { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8128       }
8129       { 1.0 }
8130     }

```

```

8131     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8132     \pgfpicture
8133     \pgfrememberpicturepositiononpagetrue
8134     \pgf@relevantforpicturesizefalse
8135     \pgftransformshift
8136     {
8137         \pgfpoint
8138         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8139         { \dim_use:N \g_tmpa_dim }
8140     }
8141     \pgfnode
8142     { rectangle }
8143     { center }
8144     { \box_use:N \l_tmpa_box }
8145     { }
8146     { }
8147     \endpgfpicture
8148 }
8149 }
8150 }

```

## 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.

```
8151 \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.

```
8152 \bool_new:N \c_@@_footnote_bool
```

```
8153 \msg_new:nnnn { nicematrix } { Unknown-key-for~package }
```

```
8154 {
8155     The~key~'\l_keys_key_str'~is-unknown. \\
8156     That~key~will~be~ignored. \\
8157     For~a~list~of~the~available~keys,~type-H~<return>.
8158 }

```

```
8159 {
8160     The~available~keys~are~(in~alphabetic~order):~
8161     footnote,~
8162     footnotehyper,~
8163     messages-for-Overleaf,~
8164     no-test-for-array,~
8165     renew-dots,~and
8166     renew-matrix.
8167 }

```

```
8168 \keys_define:nn { NiceMatrix / Package }
```

```
8169 {
8170     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8171     renew-dots .value_forbidden:n = true ,
8172     renew-matrix .code:n = \@@_renew_matrix: ,
8173     renew-matrix .value_forbidden:n = true ,
8174     messages-for-Overleaf .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
8175     footnote .bool_set:N = \c_@@_footnote_bool ,
8176     footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
8177     no-test-for-array .bool_set:N = \c_@@_no_test_for_array_bool ,
8178     no-test-for-array .default:n = true ,
8179     unknown .code:n = \@@_error:n { Unknown-key-for~package }

```

```

8180 }
8181 \ProcessKeysOptions { NiceMatrix / Package }

8182 \@@_msg_new:nn { footnote~with~footnotehyper~package }
8183 {
8184   You~can't~use~the~option~'footnote'~because~the~package~
8185   footnotehyper~has~already~been~loaded.~
8186   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
8187   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8188   of~the~package~footnotehyper.\\
8189   The~package~footnote~won't~be~loaded.
8190 }

8191 \@@_msg_new:nn { footnotehyper~with~footnote~package }
8192 {
8193   You~can't~use~the~option~'footnotehyper'~because~the~package~
8194   footnote~has~already~been~loaded.~
8195   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8196   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8197   of~the~package~footnote.\\
8198   The~package~footnotehyper~won't~be~loaded.
8199 }

8200 \bool_if:NT \c_@@_footnote_bool
8201 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8202   \@ifclassloaded { beamer }
8203     { \bool_set_false:N \c_@@_footnote_bool }
8204     {
8205       \@ifpackageloaded { footnotehyper }
8206         { \@_error:n { footnote~with~footnotehyper~package } }
8207         { \usepackage { footnote } }
8208     }
8209   }

8210 \bool_if:NT \c_@@_footnotehyper_bool
8211 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8212   \@ifclassloaded { beamer }
8213     { \bool_set_false:N \c_@@_footnote_bool }
8214     {
8215       \@ifpackageloaded { footnote }
8216         { \@_error:n { footnotehyper~with~footnote~package } }
8217         { \usepackage { footnotehyper } }
8218     }
8219   \bool_set_true:N \c_@@_footnote_bool
8220 }

```

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}`.

## About the package underscore

```

8221 \bool_new:N \l_@@_underscore_loaded_bool
8222 \@ifpackageloaded { underscore }
8223   { \bool_set_true:N \l_@@_underscore_loaded_bool }
8224   { }

8225 \hook_gput_code:nnn { begindocument } { . }

```

```

8226 {
8227   \bool_if:NF \l_@@_underscore_loaded_bool
8228   {
8229     \ifpackageloaded { underscore }
8230     { \@@_error:n { underscore~after~nicematrix } }
8231   }
8232 }

```

## Error messages of the package

```

8233 \bool_if:NTF \c_@@_messages_for_Overleaf_bool
8234 { \str_const:Nn \c_@@_available_keys_str { } }
8235 {
8236   \str_const:Nn \c_@@_available_keys_str
8237   { For~a~list~of~the~available~keys,~type~H~<return>. }
8238 }
8239 \seq_new:N \g_@@_types_of_matrix_seq
8240 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8241 {
8242   NiceMatrix ,
8243   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8244 }
8245 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8246 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in: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`.

```

8247 \cs_new_protected:Npn \@@_error_too_much_cols:
8248 {
8249   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8250   {
8251     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8252     { \@@_fatal:n { too~much~cols~for~matrix } }
8253     {
8254       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8255       { \@@_fatal:n { too~much~cols~for~matrix } }
8256       {
8257         \bool_if:NF \l_@@_last_col_without_value_bool
8258         { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
8259       }
8260     }
8261   }
8262   { \@@_fatal:n { too~much~cols~for~array } }
8263 }

```

The following command must *not* be protected since it's used in an error message.

```

8264 \cs_new:Npn \@@_message_hdotsfor:
8265 {
8266   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8267   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
8268 }
8269 \@@_msg_new:nn { negative~weight }
8270 {
8271   Negative~weight.\
8272   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
8273   the~value~'\int_use:N \l_@@_weight_int'\
8274   The~absolute~value~will~be~used.
8275 }
8276 \@@_msg_new:nn { last~col~not~used }
8277 {

```

```

8278 Column-not-used.\
8279 The-key-'last-col'-is-in-force-but-you-have-not-used-that-last-column~
8280 in-your~\@@_full_name_env:~However,~you~can~go~on.
8281 }

8282 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
8283 {
8284 Too-much-columns.\
8285 In-the-row~\int_eval:n { \c@iRow },~
8286 you-try-to-use-more-columns~
8287 than-allowed-by-your~\@@_full_name_env:.\@@_message_hdotsfor:\
8288 The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
8289 (plus-the-exterior-columns).~This-error-is-fatal.
8290 }

8291 \@@_msg_new:nn { too-much-cols-for-matrix }
8292 {
8293 Too-much-columns.\
8294 In-the-row~\int_eval:n { \c@iRow },~
8295 you-try-to-use-more-columns~than~allowed-by-your~
8296 \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
8297 number-of-columns-for-a-matrix~(excepted-the-potential-exterior~
8298 columns)-is-fixed-by-the-LaTeX-counter-'MaxMatrixCols'.~
8299 Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
8300 \token_to_str:N \setcounter\ to-change-that-value).~
8301 This-error-is-fatal.
8302 }

8303 \@@_msg_new:nn { too-much-cols-for-array }
8304 {
8305 Too-much-columns.\
8306 In-the-row~\int_eval:n { \c@iRow },~
8307 ~you-try-to-use-more-columns-than-allowed-by-your~
8308 \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
8309 \int_use:N \g_@@_static_num_of_col_int\
8310 ~(plus-the-potential-exterior-ones).~
8311 This-error-is-fatal.
8312 }

8313 \@@_msg_new:nn { columns-not-used }
8314 {
8315 Columns-not-used.\
8316 The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
8317 \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\
8318 The-columns-you-did-not-used-won't-be-created.\
8319 We-won't-have-similar-error-till-the-end-of-the-document.
8320 }

8321 \@@_msg_new:nn { in-first-col }
8322 {
8323 Erroneous-use.\
8324 You-can't-use-the-command~#1 in-the-first-column~(number-0)~of-the-array.\
8325 That-command-will-be-ignored.
8326 }

8327 \@@_msg_new:nn { in-last-col }
8328 {
8329 Erroneous-use.\
8330 You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\
8331 That-command-will-be-ignored.
8332 }

8333 \@@_msg_new:nn { in-first-row }
8334 {
8335 Erroneous-use.\
8336 You-can't-use-the-command~#1 in-the-first-row~(number-0)~of-the-array.\
8337 That-command-will-be-ignored.

```

```

8338 }
8339 \@@_msg_new:nn { in-last-row }
8340 {
8341   You~can't~use~the~command~#1~in~the~last~row~(exterior)~of~the~array.\\
8342   That~command~will~be~ignored.
8343 }
8344 \@@_msg_new:nn { caption-outside-float }
8345 {
8346   Key~caption~forbidden.\\
8347   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
8348   environment.~This~key~will~be~ignored.
8349 }
8350 \@@_msg_new:nn { short-caption-without-caption }
8351 {
8352   You~should~not~use~the~key~'short-caption'~without~'caption'.~
8353   However,~your~'short-caption'~will~be~used~as~'caption'.
8354 }
8355 \@@_msg_new:nn { double-closing-delimiter }
8356 {
8357   Double~delimiter.\\
8358   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
8359   delimiter.~This~delimiter~will~be~ignored.
8360 }
8361 \@@_msg_new:nn { delimiter-after-opening }
8362 {
8363   Double~delimiter.\\
8364   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
8365   delimiter.~That~delimiter~will~be~ignored.
8366 }
8367 \@@_msg_new:nn { bad-option-for-line-style }
8368 {
8369   Bad~line~style.\\
8370   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
8371   is~'standard'.~That~key~will~be~ignored.
8372 }
8373 \@@_msg_new:nn { Identical-notes-in-caption }
8374 {
8375   Identical~tabular~notes.\\
8376   You~can't~put~several~notes~with~the~same~content~in~
8377   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
8378   If~you~go~on,~the~output~will~probably~be~erroneous.
8379 }
8380 \@@_msg_new:nn { tabularnote-below-the-tabular }
8381 {
8382   \token_to_str:N \tabularnote\ forbidden\\
8383   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
8384   of~your~tabular~because~the~caption~will~be~composed~below~
8385   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
8386   key~'caption-above'~in~\token_to_str:N \NiceMatrixOptions.\\
8387   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
8388   no~similar~error~will~raised~in~this~document.
8389 }
8390 \@@_msg_new:nn { Unknown-key-for-rules }
8391 {
8392   Unknown~key.\\
8393   There~is~only~two~keys~available~here:~width~and~color.\\
8394   You~key~'\l_keys_key_str'~will~be~ignored.
8395 }
8396 \@@_msg_new:nnn { Unknown-key-for-custom-line }

```

```

8397 {
8398   Unknown~key.\\
8399   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom-line'.~
8400   It~you~go~on,~you~will~probably~have~other~errors. \\
8401   \c_@@_available_keys_str
8402 }
8403 {
8404   The~available~keys~are~(in~alphabetic~order):~
8405   ccommand,~
8406   color,~
8407   command,~
8408   dotted,~
8409   letter,~
8410   multiplicity,~
8411   sep-color,~
8412   tikz,~and~total-width.
8413 }
8414 \@@_msg_new:nnn { Unknown~key~for~xdots }
8415 {
8416   Unknown~key.\\
8417   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
8418   \c_@@_available_keys_str
8419 }
8420 {
8421   The~available~keys~are~(in~alphabetic~order):~
8422   'color',~
8423   'inter',~
8424   'line-style',~
8425   'radius',~
8426   'shorten',~
8427   'shorten-end'~and~'shorten-start'.
8428 }
8429 \@@_msg_new:nn { Unknown~key~for~rowcolors }
8430 {
8431   Unknown~key.\\
8432   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
8433   (and~you~try~to~use~'\l_keys_key_str')\\
8434   That~key~will~be~ignored.
8435 }
8436 \@@_msg_new:nn { label~without~caption }
8437 {
8438   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
8439   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
8440 }
8441 \@@_msg_new:nn { W~warning }
8442 {
8443   Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
8444   (row~\int_use:N \c@iRow).
8445 }
8446 \@@_msg_new:nn { Construct~too~large }
8447 {
8448   Construct~too~large.\\
8449   Your~command~\token_to_str:N #1
8450   can't~be~drawn~because~your~matrix~is~too~small.\\
8451   That~command~will~be~ignored.
8452 }
8453 \@@_msg_new:nn { underscore~after~nicematrix }
8454 {
8455   Problem~with~'underscore'.\\
8456   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
8457   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\

```



```

8458     '\token_to_str:N \Cdots\token_to_str:N _{n-\token_to_str:N \text{\~times}}}'.
8459   }
8460 \@@_msg_new:nn { ampersand-in-light-syntax }
8461   {
8462     Ampersand-forbidden.\
8463     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
8464     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
8465   }
8466 \@@_msg_new:nn { double-backslash-in-light-syntax }
8467   {
8468     Double-backslash-forbidden.\
8469     You~can't~use~\token_to_str:N
8470     \~to~separate~rows~because~the~key~'light-syntax'~
8471     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
8472     (set~by~the~key~'end-of-row').~This~error~is~fatal.
8473   }
8474 \@@_msg_new:nn { hlines-with-color }
8475   {
8476     Incompatible~keys.\
8477     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
8478     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\
8479     Maybe~it~will~be~possible~in~future~version.\
8480     Your~key~will~be~discarded.
8481   }
8482 \@@_msg_new:nn { bad-value-for-baseline }
8483   {
8484     Bad~value~for~baseline.\
8485     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
8486     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
8487     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
8488     the~form~'line-i'.\
8489     A~value~of~1~will~be~used.
8490   }
8491 \@@_msg_new:nn { ragged2e-not-loaded }
8492   {
8493     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
8494     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
8495     \l_keys_key_str'~will~be~used~instead.
8496   }
8497 \@@_msg_new:nn { Invalid-name }
8498   {
8499     Invalid~name.\
8500     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
8501     \SubMatrix\ of~your~\@@_full_name_env:.\
8502     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\
8503     This~key~will~be~ignored.
8504   }
8505 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
8506   {
8507     Wrong~line.\
8508     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
8509     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
8510     number~is~not~valid.~It~will~be~ignored.
8511   }
8512 \@@_msg_new:nn { Impossible-delimiter }
8513   {
8514     Impossible~delimiter.\
8515     It's~impossible~to~draw~the~#1~delimiter~of~your~
8516     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
8517     in~that~column.

```

```

8518 \bool_if:NT \l_@@_submatrix_slim_bool
8519   { ~Maybe-you-should-try-without-the-key~'slim'. } \\
8520   This~\token_to_str:N \SubMatrix\ will~be~ignored.
8521 }

8522 \@@_msg_new:nn { width-without-X-columns }
8523 {
8524   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
8525   That~key~will~be~ignored.
8526 }

8527 \@@_msg_new:nn { key-multiplicity-with-dotted }
8528 {
8529   Incompatible~keys. \\
8530   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
8531   in~a~'custom-line'.~They~are~incompatible. \\
8532   The~key~'multiplicity'~will~be~discarded.
8533 }

8534 \@@_msg_new:nn { empty-environment }
8535 {
8536   Empty~environment.\\
8537   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
8538 }

8539 \@@_msg_new:nn { No-letter-and-no-command }
8540 {
8541   Erroneous~use.\\
8542   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
8543   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
8544   '~'ccommand'~(to~draw~horizontal~rules).\\
8545   However,~you~can~go~on.
8546 }

8547 \@@_msg_new:nn { Forbidden-letter }
8548 {
8549   Forbidden~letter.\\
8550   You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
8551   It~will~be~ignored.
8552 }

8553 \@@_msg_new:nn { Several-letters }
8554 {
8555   Wrong~name.\\
8556   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
8557   have~used~'\l_@@_letter_str').\\
8558   It~will~be~ignored.
8559 }

8560 \@@_msg_new:nn { Delimiter-with-small }
8561 {
8562   Delimiter~forbidden.\\
8563   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
8564   because~the~key~'small'~is~in~force.\\
8565   This~error~is~fatal.
8566 }

8567 \@@_msg_new:nn { unknown-cell-for-line-inCodeAfter }
8568 {
8569   Unknown~cell.\\
8570   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
8571   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
8572   can't~be~executed~because~a~cell~doesn't~exist.\\
8573   This~command~\token_to_str:N \line\ will~be~ignored.
8574 }

8575 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
8576 {
8577   Duplicate~name.\\

```

```

8578 The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
8579 in~this~\@@_full_name_env:.\
8580 This~key~will~be~ignored.\
8581 \bool_if:NF \c_@@_messages_for_Overleaf_bool
8582 { For~a~list~of~the~names~already~used,~type~H~<return>. }
8583 }
8584 {
8585 The~names~already~defined~in~this~\@@_full_name_env:\ are:~
8586 \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
8587 }
8588 \@@_msg_new:nn { r~or~l~with~preamble }
8589 {
8590 Erroneous~use.\
8591 You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
8592 You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
8593 your~\@@_full_name_env:.\
8594 This~key~will~be~ignored.
8595 }
8596 \@@_msg_new:nn { Hdotsfor~in~col~0 }
8597 {
8598 Erroneous~use.\
8599 You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
8600 the~array.~This~error~is~fatal.
8601 }
8602 \@@_msg_new:nn { bad~corner }
8603 {
8604 Bad~corner.\
8605 #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
8606 'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\
8607 This~specification~of~corner~will~be~ignored.
8608 }
8609 \@@_msg_new:nn { bad~border }
8610 {
8611 Bad~border.\
8612 \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
8613 (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
8614 The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
8615 also~use~the~key~'tikz'
8616 \bool_if:nF \c_@@_tikz_loaded_bool
8617 {~if~you~load~the~LaTeX~package~'tikz'}).\
8618 This~specification~of~border~will~be~ignored.
8619 }
8620 \@@_msg_new:nn { tikz~key~without~tikz }
8621 {
8622 Tikz~not~loaded.\
8623 You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
8624 \Block'~because~you~have~not~loaded~tikz.~
8625 This~key~will~be~ignored.
8626 }
8627 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
8628 {
8629 Erroneous~use.\
8630 In~the~\@@_full_name_env:,~you~must~use~the~key~
8631 'last~col'~without~value.\
8632 However,~you~can~go~on~for~this~time~
8633 (the~value~'\l_keys_value_tl'~will~be~ignored).
8634 }
8635 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
8636 {
8637 Erroneous~use.\
8638 In~\NiceMatrixoptions,~you~must~use~the~key~

```

```

8639 'last-col'~without~value.\\
8640 However,~you~can~go~on~for~this~time~
8641 (the~value~'\l_keys_value_tl'~will~be~ignored).
8642 }

8643 \@@_msg_new:nn { Block-too-large-1 }
8644 {
8645   Block-too-large.\\
8646   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
8647   too~small~for~that~block. \\
8648 }

8649 \@@_msg_new:nn { Block-too-large-2 }
8650 {
8651   Block-too-large.\\
8652   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8653   \g_@@_static_num_of_col_int\
8654   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
8655   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
8656   (&)~at~the~end~of~the~first~row~of~your~
8657   \@@_full_name_env:.\\
8658   This~block~and~maybe~others~will~be~ignored.
8659 }

8660 \@@_msg_new:nn { unknown-column-type }
8661 {
8662   Bad~column~type.\\
8663   The~column~type~'#1'~in~your~\@@_full_name_env:\
8664   is~unknown. \\
8665   This~error~is~fatal.
8666 }

8667 \@@_msg_new:nn { tabularnote-forbidden }
8668 {
8669   Forbidden~command.\\
8670   You~can't~use~the~command~\token_to_str:N\tabularnote\
8671   ~here.~This~command~is~available~only~in~
8672   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
8673   the~argument~of~a~command~\token_to_str:N \caption\ included~
8674   in~an~environment~{table}. \\
8675   This~command~will~be~ignored.
8676 }

8677 \@@_msg_new:nn { borders-forbidden }
8678 {
8679   Forbidden~key.\\
8680   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
8681   because~the~option~'rounded-corners'~
8682   is~in~force~with~a~non-zero~value.\\
8683   This~key~will~be~ignored.
8684 }

8685 \@@_msg_new:nn { bottomrule-without-booktabs }
8686 {
8687   booktabs~not~loaded.\\
8688   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
8689   loaded~'booktabs'.\\
8690   This~key~will~be~ignored.
8691 }

8692 \@@_msg_new:nn { enumitem-not-loaded }
8693 {
8694   enumitem~not~loaded.\\
8695   You~can't~use~the~command~\token_to_str:N\tabularnote\
8696   ~because~you~haven't~loaded~'enumitem'.\\
8697   All~the~commands~\token_to_str:N\tabularnote\ will~be~
8698   ignored~in~the~document.
8699 }

```

```

8700 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
8701 {
8702   Tikz-not-loaded.\\
8703   You-have-used-the-key-'tikz'-in-the-definition-of-a-
8704   customized-line-(with-'custom-line')~but~tikz-is-not-loaded.~
8705   You-can-go-on-but-you-will-have-another-error-if-you-actually-
8706   use-that-custom-line.
8707 }
8708 \@@_msg_new:nn { tikz-in-borders-without-tikz }
8709 {
8710   Tikz-not-loaded.\\
8711   You-have-used-the-key-'tikz'-in-a-key-'borders'-(of-a-
8712   command'\token_to_str:N\Block')~but~tikz-is-not-loaded.~
8713   That-key-will-be-ignored.
8714 }
8715 \@@_msg_new:nn { color-in-custom-line-with-tikz }
8716 {
8717   Erroneous-use.\\
8718   In-a-'custom-line',-you-have-used-both-'tikz'~and-'color',~
8719   which-is-forbidden-(you-should-use-'color'-inside-the-key-'tikz').~
8720   The-key-'color'-will-be-discarded.
8721 }
8722 \@@_msg_new:nn { Wrong-last-row }
8723 {
8724   Wrong-number.\\
8725   You-have-used-'last-row=\int_use:N \l_@@_last_row_int'-but-your-
8726   \@@_full_name_env:\ seems-to-have-\int_use:N \c@iRow \ rows.~
8727   If-you-go-on,~the-value-of-\int_use:N \c@iRow \ will-be-used-for-
8728   last-row.~You-can-avoid-this-problem-by-using-'last-row'~
8729   without-value-(more-compilations-might-be-necessary).
8730 }
8731 \@@_msg_new:nn { Yet-in-env }
8732 {
8733   Nested-environments.\\
8734   Environments-of-nicematrix-can't-be-nested.\\
8735   This-error-is-fatal.
8736 }
8737 \@@_msg_new:nn { Outside-math-mode }
8738 {
8739   Outside-math-mode.\\
8740   The-\@@_full_name_env:\ can-be-used-only-in-math-mode-
8741   (and-not-in-\token_to_str:N \vcenter).\\
8742   This-error-is-fatal.
8743 }
8744 \@@_msg_new:nn { One-letter-allowed }
8745 {
8746   Bad-name.\\
8747   The-value-of-key-\l_keys_key_str'-must-be-of-length-1.\\
8748   It-will-be-ignored.
8749 }
8750 \@@_msg_new:nn { TabularNote-in-CodeAfter }
8751 {
8752   Environment~{TabularNote}~forbidden.\\
8753   You-must-use~{TabularNote}~at-the-end-of-your~{NiceTabular}~
8754   but~*before*~the-\token_to_str:N \CodeAfter.\\
8755   This-environment~{TabularNote}~will-be-ignored.
8756 }
8757 \@@_msg_new:nn { varwidth-not-loaded }
8758 {
8759   varwidth-not-loaded.\\

```

```

8760   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
8761   loaded.\\
8762   Your~column~will~behave~like~'p'.
8763   }
8764   \@@_msg_new:nnn { Unknown~key~for~RulesBis }
8765   {
8766     Unknow~key.\\
8767     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
8768     \c_@@_available_keys_str
8769   }
8770   {
8771     The~available~keys~are~(in~alphabetic~order):~
8772     color,~
8773     dotted,~
8774     multiplicity,~
8775     sep-color,~
8776     tikz,~and~total-width.
8777   }
8778
8779   \@@_msg_new:nnn { Unknown~key~for~Block }
8780   {
8781     Unknown~key.\\
8782     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
8783     \Block.\\ It~will~be~ignored. \\
8784     \c_@@_available_keys_str
8785   }
8786   {
8787     The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
8788     hlines,~hvlines,~l,~line-width,~name,~rounded-corners,~r,~respect-arraystretch,~
8789     t,~T,~tikz,~transparent~and~vlines.
8790   }
8791   \@@_msg_new:nn { Version~of~siunitx~too~old }
8792   {
8793     siunitx~too~old.\\
8794     You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
8795     is~too~old.~You~need~at~least~v3.0~and~your~log~file~says:~"siunitx,~
8796     \use:c { ver @ siunitx.sty }". \\
8797     This~error~is~fatal.
8798   }
8799   \@@_msg_new:nnn { Unknown~key~for~Brace }
8800   {
8801     Unknown~key.\\
8802     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
8803     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
8804     It~will~be~ignored. \\
8805     \c_@@_available_keys_str
8806   }
8807   {
8808     The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
8809     right-shorten,~shorten~(which~fixes~both~left-shorten~and~
8810     right-shorten)~and~yshift.
8811   }
8812   \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
8813   {
8814     Unknown~key.\\
8815     The~key~'\l_keys_key_str'~is~unknown.\\
8816     It~will~be~ignored. \\
8817     \c_@@_available_keys_str
8818   }
8819   {
8820     The~available~keys~are~(in~alphabetic~order):~
8821     delimiters/color,~

```

```

8822 rules~(with~the~subkeys~'color'~and~'width'),~
8823 sub-matrix~(several~subkeys)~
8824 and~xdots~(several~subkeys).~
8825 The~latter~is~for~the~command~\token_to_str:N \line.
8826 }
8827 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
8828 {
8829   Unknown~key.\\
8830   The~key~'\l_keys_key_str'~is~unknown.\\
8831   It~will~be~ignored. \\
8832   \c_@@_available_keys_str
8833 }
8834 {
8835   The~available~keys~are~(in~alphabetic~order):~
8836   create-cell-nodes,~
8837   delimiters/color~and~
8838   sub-matrix~(several~subkeys).
8839 }
8840 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
8841 {
8842   Unknown~key.\\
8843   The~key~'\l_keys_key_str'~is~unknown.\\
8844   That~key~will~be~ignored. \\
8845   \c_@@_available_keys_str
8846 }
8847 {
8848   The~available~keys~are~(in~alphabetic~order):~
8849   'delimiters/color',~
8850   'extra-height',~
8851   'hlines',~
8852   'hvlines',~
8853   'left-xshift',~
8854   'name',~
8855   'right-xshift',~
8856   'rules'~(with~the~subkeys~'color'~and~'width'),~
8857   'slim',~
8858   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
8859   and~'right-xshift').\\
8860 }
8861 \@@_msg_new:nnn { Unknown~key~for~notes }
8862 {
8863   Unknown~key.\\
8864   The~key~'\l_keys_key_str'~is~unknown.\\
8865   That~key~will~be~ignored. \\
8866   \c_@@_available_keys_str
8867 }
8868 {
8869   The~available~keys~are~(in~alphabetic~order):~
8870   bottomrule,~
8871   code-after,~
8872   code-before,~
8873   detect-duplicates,~
8874   enumitem-keys,~
8875   enumitem-keys-para,~
8876   para,~
8877   label-in-list,~
8878   label-in-tabular~and~
8879   style.
8880 }
8881 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
8882 {
8883   Unknown~key.\\

```

```

8884 The~key~'\l_keys_key_str'~is~unknown~for~the~command~
8885 \token_to_str:N \RowStyle. \\
8886 That~key~will~be~ignored. \\
8887 \c_@@_available_keys_str
8888 }
8889 {
8890 The~available~keys~are~(in~alphabetic~order):~
8891 'bold',~
8892 'cell-space-top-limit',~
8893 'cell-space-bottom-limit',~
8894 'cell-space-limits',~
8895 'color',~
8896 'nb-rows'~and~
8897 'rowcolor'.
8898 }

8899 \@@_msg_new:nnn { Unknown-key~for~NiceMatrixOptions }
8900 {
8901 Unknown~key.\\
8902 The~key~'\l_keys_key_str'~is~unknown~for~the~command~
8903 \token_to_str:N \NiceMatrixOptions. \\
8904 That~key~will~be~ignored. \\
8905 \c_@@_available_keys_str
8906 }
8907 {
8908 The~available~keys~are~(in~alphabetic~order):~
8909 allow-duplicate-names,~
8910 caption-above,~
8911 cell-space-bottom-limit,~
8912 cell-space-limits,~
8913 cell-space-top-limit,~
8914 code-for-first-col,~
8915 code-for-first-row,~
8916 code-for-last-col,~
8917 code-for-last-row,~
8918 corners,~
8919 custom-key,~
8920 create-extra-nodes,~
8921 create-medium-nodes,~
8922 create-large-nodes,~
8923 delimiters~(several~subkeys),~
8924 end-of-row,~
8925 first-col,~
8926 first-row,~
8927 hlines,~
8928 hvlines,~
8929 last-col,~
8930 last-row,~
8931 left-margin,~
8932 light-syntax,~
8933 matrix/columns-type,~
8934 notes~(several~subkeys),~
8935 nullify-dots,~
8936 renew-dots,~
8937 renew-matrix,~
8938 respect-arraystretch,~
8939 right-margin,~
8940 rules~(with~the~subkeys~'color'~and~'width'),~
8941 small,~
8942 sub-matrix~(several~subkeys),~
8943 vlines,~
8944 xdots~(several~subkeys).
8945 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and



r.

```
8946 \@_msg_new:nnn { Unknown~key~for~NiceArray }
8947   {
8948     Unknown~key.\\
8949     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
8950     \{NiceArray\}. \\
8951     That~key~will~be~ignored. \\
8952     \c_@@_available_keys_str
8953   }
8954   {
8955     The~available~keys~are~(in~alphabetic~order):~
8956     b,~
8957     baseline,~
8958     c,~
8959     cell-space-bottom-limit,~
8960     cell-space-limits,~
8961     cell-space-top-limit,~
8962     code-after,~
8963     code-for-first-col,~
8964     code-for-first-row,~
8965     code-for-last-col,~
8966     code-for-last-row,~
8967     colortbl-like,~
8968     columns-width,~
8969     corners,~
8970     create-extra-nodes,~
8971     create-medium-nodes,~
8972     create-large-nodes,~
8973     extra-left-margin,~
8974     extra-right-margin,~
8975     first-col,~
8976     first-row,~
8977     hlines,~
8978     hvlines,~
8979     last-col,~
8980     last-row,~
8981     left-margin,~
8982     light-syntax,~
8983     name,~
8984     nullify-dots,~
8985     renew-dots,~
8986     respect-arraystretch,~
8987     right-margin,~
8988     rules~(with~the~subkeys~'color'~and~'width'),~
8989     small,~
8990     t,~
8991     tabulernote,~
8992     vlines,~
8993     xdots/color,~
8994     xdots/shorten-start,~
8995     xdots/shorten-end,~
8996     xdots/shorten-and~
8997     xdots/line-style.
8998   }
```

This error message is used for the set of keys NiceMatrix/NiceMatrix and NiceMatrix/pNiceArray (but not by NiceMatrix/NiceArray because, for this set of keys, there is no l and r).

```
8999 \@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9000   {
9001     Unknown~key.\\
9002     The~key~'\l_keys_key_str'~is~unknown~for~the~
9003     \@_full_name_env:. \\
9004     That~key~will~be~ignored. \\
9005     \c_@@_available_keys_str
```

```

9006 }
9007 {
9008   The~available~keys~are~(in~alphabetic~order):~
9009   b,~
9010   baseline,~
9011   c,~
9012   cell-space-bottom-limit,~
9013   cell-space-limits,~
9014   cell-space-top-limit,~
9015   code-after,~
9016   code-for-first-col,~
9017   code-for-first-row,~
9018   code-for-last-col,~
9019   code-for-last-row,~
9020   colortbl-like,~
9021   columns-type,~
9022   columns-width,~
9023   corners,~
9024   create-extra-nodes,~
9025   create-medium-nodes,~
9026   create-large-nodes,~
9027   extra-left-margin,~
9028   extra-right-margin,~
9029   first-col,~
9030   first-row,~
9031   hlines,~
9032   hvlines,~
9033   l,~
9034   last-col,~
9035   last-row,~
9036   left-margin,~
9037   light-syntax,~
9038   name,~
9039   nullify-dots,~
9040   r,~
9041   renew-dots,~
9042   respect-arraystretch,~
9043   right-margin,~
9044   rules~(with~the~subkeys~'color'~and~'width'),~
9045   small,~
9046   t,~
9047   vlines,~
9048   xdots/color,~
9049   xdots/shorten-start,~
9050   xdots/shorten-end,~
9051   xdots/shorten~and~
9052   xdots/line-style.
9053 }
9054 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9055 {
9056   Unknown~key.\\
9057   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9058   \{NiceTabular\}. \\
9059   That~key~will~be~ignored. \\
9060   \c_@@_available_keys_str
9061 }
9062 {
9063   The~available~keys~are~(in~alphabetic~order):~
9064   b,~
9065   baseline,~
9066   c,~
9067   caption,~
9068   cell-space-bottom-limit,~

```

```

9069 cell-space-limits,~
9070 cell-space-top-limit,~
9071 code-after,~
9072 code-for-first-col,~
9073 code-for-first-row,~
9074 code-for-last-col,~
9075 code-for-last-row,~
9076 colortbl-like,~
9077 columns-width,~
9078 corners,~
9079 custom-line,~
9080 create-extra-nodes,~
9081 create-medium-nodes,~
9082 create-large-nodes,~
9083 extra-left-margin,~
9084 extra-right-margin,~
9085 first-col,~
9086 first-row,~
9087 hlines,~
9088 hvlines,~
9089 label,~
9090 last-col,~
9091 last-row,~
9092 left-margin,~
9093 light-syntax,~
9094 name,~
9095 notes~(several~subkeys),~
9096 nullify-dots,~
9097 renew-dots,~
9098 respect-arraystretch,~
9099 right-margin,~
9100 rules~(with~the~subkeys~'color'~and~'width'),~
9101 short-caption,~
9102 t,~
9103 tabularnote,~
9104 vlines,~
9105 xdots/color,~
9106 xdots/shorten-start,~
9107 xdots/shorten-end,~
9108 xdots/shorten~and~
9109 xdots/line-style.
9110 }

9111 \@@_msg_new:nnn { Duplicate-name }
9112 {
9113 Duplicate-name.\
9114 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9115 the~same~environment~name~twice.~You~can~go~on,~but,~
9116 maybe,~you~will~have~incorrect~results~especially~
9117 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9118 message~again,~use~the~key~'allow-duplicate-names'~in~
9119 '\token_to_str:N \NiceMatrixOptions'.\
9120 \c_@@_available_keys_str
9121 }
9122 {
9123 The~names~already~defined~in~this~document~are:~
9124 \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9125 }

9126 \@@_msg_new:nn { Option~auto~for~columns~width }
9127 {
9128 Erroneous~use.\
9129 You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9130 That~key~will~be~ignored.
9131 }

```

## 20 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 version 6.14 and 6.15

New key `transparent` for the command `\Block` (with that key, the rules are drawn within the block).

### Changes between version 6.13 and 6.14

New keys for the command `\Block` for the vertical position of the content of that block.

### Changes between version 6.12 and 6.13

New environment `{TabularNote}` in `{NiceTabular}` with the same semantic as the key `tabularnote` (for legibility).

The command `\Hline` now accepts options (between square brackets).

### Changes between version 6.11 and 6.12

New keys `caption`, `short-caption` and `label` in the environment `{NiceTabular}`.

In `{NiceTabular}`, a caption specified by the key `caption` is wrapped to the width of the tabular.

Correction of a bug: it's now possible to use `\OverBrace` and `\UnderBrace` with `unicode-math` (with XeLaTeX or LuaLaTeX).

### Changes between version 6.10 and 6.11

New key `matrix/columns-type` to specify the type of columns of the matrices.

New key `ccommand` in `custom-line` and new command `\cdottedline`.

### Changes between version 6.9 and 6.10

New keys `xdots/shorten-start` and `xdots/shorten-end`.

It's possible to use `\line` in the `\CodeAfter` between two blocks (and not only two cells).

### Changes between version 6.8 and 6.9

New keys `xdots/radius` and `xdots/inter` for customisation of the continuous dotted lines.

New command `\ShowCellNames` available in the `\CodeBefore` and in the `\CodeAfter`.

### Changes between version 6.7 and 6.8

In the notes of a tabular (with the command `\tabularnote`), the duplicates are now detected: when several commands `\tabularnote` are used with the same argument, only one note is created at the end of the tabular (but all the labels are present, of course).

### Changes between version 6.6 and 6.7

Key `color` for `\OverBrace` and `\UnderBrace` in the `\CodeAfter`

Key `tikz` in the key `borders` of a command `\Block`

## Changes between version 6.5 and 6.6

Keys `tikz` and `width` in `custom-line`.

## Changes between versions 6.4 and 6.5

Key `custom-line` in `\NiceMatrixOptions`.

Key `respect-arraystretch`.

## Changes between versions 6.3 and 6.4

New commands `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

Correction of a bug of the key `baseline` (cf. question 623258 on TeX StackExchange).

Correction of a bug with the columns `V` of `varwidth`.

Correction of a bug: the use of `\hdottedline` and `:` in the preamble of the array (of another letter specified by `letter-for-dotted-lines`) was incompatible with the key `xdots/line-style`.

## Changes between versions 6.2 and 6.3

Keys `nb-rows`, `rowcolor` and `bold` for the command `\RowStyle`

Key `name` for the command `\Block`.

Support for the columns `V` of `varwidth`.

## Changes between versions 6.1 and 6.2

Better compatibility with the classes `revtex4-1` and `revtex4-2`.

Key `vlines-in-sub-matrix`.

## Changes between versions 6.0 and 6.1

Better computation of the widths of the `X` columns.

Key `\color` for the command `\RowStyle`.

## Changes between versions 5.19 and 6.0

Columns `X` and environment `{NiceTabularX}`.

Command `\rowlistcolors` available in the `\CodeBefore`.

In columns with fixed width, the blocks are composed as paragraphs (wrapping of the lines).

The key `define-L-C-R` has been deleted.

## Changes between versions 5.18 and 5.19

New key `tikz` for the command `\Block`.

## Changes between versions 5.17 and 5.18

New command `\RowStyle`

## Changes between versions 5.16 and 5.17

The key `define-L-C-R` (only available at load-time) now raises a (non fatal) error.

Keys `L`, `C` and `R` for the command `\Block`.

Key `hvlines-except-borders`.

It's now possible to use a key `l`, `r` or `c` with the command `\pAutoNiceMatrix` (and the similar ones).

## Changes between versions 5.15 and 5.16

It's now possible to use the cells corresponding to the contents of the nodes (of the form  $i-j$ ) in the `\CodeBefore` when the key `create-cell-nodes` of that `\CodeBefore` is used. The medium and the large nodes are also available if the corresponding keys are used.

## Changes between versions 5.14 and 5.15

Key `hvlines` for the command `\Block`.

The commands provided by `nicematrix` to color cells, rows and columns don't color the cells which are in the "corners" (when the key `corner` is used).

It's now possible to specify delimiters for submatrices in the preamble of an environment.

The version 5.15b is compatible with the version 3.0+ of `siunitx` (previous versions were not).

## Changes between versions 5.13 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 5.12 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 5.11 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 5.10 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 5.9 and 5.10

New command `\SubMatrix` available in the `\CodeAfter`.

It's possible to provide options (between brackets) to the keyword `\CodeAfter`.

## 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 5.7 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 5.6 and 5.7

New key `delimiters-color`

Keys `fill`, `draw` and `line-width` for the command `\Block`.

## Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.

New command `\NotEmpty`.

## 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 5.3 and 5.4

Key `tabularnote`.

Different behaviour for the mono-column blocks.

## 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 5.1 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 5.0 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 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 4.3 and 4.4

New key `hvlines-except-corners` (now deprecated).

## 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 4.1 and 4.2

It's now possible to write `\begin{pNiceMatrix}a&b\c&d\end{pNiceMatrix}`<sup>2</sup> with the expected result.

## Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`

New command `\diagbox`

The key `hvlines` don't draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

## 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 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 3.13 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 3.12 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 3.11 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 3.10 and 3.11

Correction of a bug linked to `first-row` and `last-row`.

## 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 version 3.8 and 3.9

New commands `\NiceMatrixLastEnv` and `\OnlyMainNiceMatrix`.

New options `create-medium-nodes` and `create-large-nodes`.

## 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 3.6 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 3.5 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 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

## Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange<sup>80</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 3.2 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 previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

## Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

## 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 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 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.

Option `hlines`.

## Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “:” in the preamble.

Modification of the position of the dotted lines drawn by `\hdottedline`.

## Changes between version 2.1.5 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`).

---

<sup>80</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 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.  
Option `allow-duplicate-names`.

## Changes between version 2.1.3 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 2.1.2 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`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & \dot{\cdot} \cdots \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

## 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>81</sup>, Tikz externalization is now deactivated in the environments of the package `nicematrix`.<sup>82</sup>

## 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 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 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.

---

<sup>81</sup>cf. [tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package](http://tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package)

<sup>82</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 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 versions 1.1 and 1.2**

New environment `{NiceArray}` with column types L, C and R.

### **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.

# Contents

|           |  |           |
|-----------|--|-----------|
| <b>1</b>  | <b>The environments of this package</b>  | <b>2</b>  |
| <b>2</b>  | <b>The vertical space between the rows</b>   | <b>2</b>  |
| <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 . . . . .   | 6         |
| 4.3       | The mono-row blocks . . . . .  | 6         |
| 4.4       | The mono-cell blocks . . . . .   | 6         |
| 4.5       | Horizontal position of the content of the block . . . . .  | 7         |
| 4.6       | Vertical position of the content of the block . . . . .  | 8         |
| <b>5</b>  | <b>The rules</b>   | <b>9</b>  |
| 5.1       | Some differences with the classical environments . . . . .   | 9         |
| 5.1.1     | The vertical rules . . . . .   | 9         |
| 5.1.2     | The command <code>\cline</code> . . . . .  | 10        |
| 5.2       | The thickness and the color of the rules . . . . .   | 10        |
| 5.3       | The tools of nicematrix for the rules . . . . .  | 11        |
| 5.3.1     | The keys <code>hlines</code> and <code>vlines</code> . . . . .   | 11        |
| 5.3.2     | The keys <code>hvlines</code> and <code>hvlines-except-borders</code> . . . . .                            | 11        |
| 5.3.3     | The (empty) corners . . . . .  | 12        |
| 5.4       | The command <code>\diagbox</code> . . . . .  | 13        |
| 5.5       | Commands for customized rules . . . . .  | 13        |
| <b>6</b>  | <b>The color of the rows and columns</b>   | <b>15</b> |
| 6.1       | Use of <code>colortbl</code> . . . . .   | 15        |
| 6.2       | The tools of nicematrix in the <code>\CodeBefore</code> . . . . .  | 16        |
| 6.3       | Color tools with the syntax of <code>colortbl</code> . . . . .   | 20        |
| <b>7</b>  | <b>The command <code>\RowStyle</code></b>  | <b>20</b> |
| <b>8</b>  | <b>The width of the columns</b>  | <b>21</b> |
| 8.1       | Basic tools . . . . .  | 21        |
| 8.2       | The columns <code>V</code> of <code>varwidth</code> . . . . .  | 22        |
| 8.3       | The columns <code>X</code> . . . . .   | 23        |
| <b>9</b>  | <b>The exterior rows and columns</b>   | <b>24</b> |
| <b>10</b> | <b>The continuous dotted lines</b>   | <b>25</b> |
| 10.1      | The option <code>nullify-dots</code> . . . . .   | 27        |
| 10.2      | The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code> . . . . .                                   | 27        |
| 10.3      | How to generate the continuous dotted lines transparently . . . . .  | 28        |
| 10.4      | The labels of the dotted lines . . . . .   | 29        |
| 10.5      | Customisation of the dotted lines . . . . .  | 29        |
| 10.6      | The dotted lines and the rules . . . . .   | 30        |
| <b>11</b> | <b>The <code>\CodeAfter</code></b>   | <b>31</b> |
| 11.1      | The command <code>\line</code> in the <code>\CodeAfter</code> . . . . .                                    | 31        |
| 11.2      | The command <code>\SubMatrix</code> in the <code>\CodeAfter</code> . . . . .                               | 32        |
| 11.3      | The commands <code>\OverBrace</code> and <code>\UnderBrace</code> in the <code>\CodeAfter</code> . . . . . | 34        |

|           |   |            |
|-----------|---|------------|
| <b>12</b> | <b>Captions and notes in the tabulars</b>   | <b>35</b>  |
| 12.1      | Caption of a tabular . . . . .  | 35         |
| 12.2      | The footnotes . . . . .   | 35         |
| 12.3      | The notes of tabular . . . . .  | 35         |
| 12.4      | Customisation of the tabular notes . . . . .  | 37         |
| 12.5      | Use of <code>{NiceTabular}</code> with <code>threeparttable</code> . . . . .            | 39         |
| <b>13</b> | <b>Other features</b>   | <b>39</b>  |
| <b>14</b> | <b>Autres fonctionnalités</b>   | <b>39</b>  |
| 14.1      | Command <code>\ShowCellNames</code> . . . . .   | 39         |
| 14.2      | Use of the column type <code>S</code> of <code>siunitx</code> . . . . .                 | 40         |
| 14.3      | Default column type in <code>{NiceMatrix}</code> . . . . .                              | 40         |
| 14.4      | The command <code>\rotate</code> . . . . .  | 40         |
| 14.5      | The option <code>small</code> . . . . .   | 41         |
| 14.6      | The counters <code>iRow</code> and <code>jCol</code> . . . . .                          | 41         |
| 14.7      | The key <code>light-syntax</code> . . . . .   | 42         |
| 14.8      | Color of the delimiters . . . . .   | 42         |
| 14.9      | The environment <code>{NiceArrayWithDelims}</code> . . . . .                            | 43         |
| 14.10     | The command <code>\OnlyMainNiceMatrix</code> . . . . .                                  | 43         |
| <b>15</b> | <b>Use of Tikz with <code>nicematrix</code></b>   | <b>43</b>  |
| 15.1      | The nodes corresponding to the contents of the cells . . . . .                          | 43         |
| 15.1.1    | The columns <code>V</code> of <code>varwidth</code> . . . . .                           | 44         |
| 15.2      | The medium nodes and the large nodes . . . . .  | 44         |
| 15.3      | The nodes which indicate the position of the rules . . . . .                            | 46         |
| 15.4      | The nodes corresponding to the command <code>\SubMatrix</code> . . . . .                | 47         |
| <b>16</b> | <b>API for the developpers</b>  | <b>48</b>  |
| <b>17</b> | <b>Technical remarks</b>  | <b>49</b>  |
| 17.1      | Diagonal lines . . . . .  | 49         |
| 17.2      | The empty cells . . . . .   | 49         |
| 17.3      | The option <code>exterior-arraycolsep</code> . . . . .                                  | 50         |
| 17.4      | Incompatibilities . . . . .   | 50         |
| <b>18</b> | <b>Examples</b>   | <b>51</b>  |
| 18.1      | Utilisation of the key <code>'tikz'</code> of the command <code>\Block</code> . . . . . | 51         |
| 18.2      | Notes in the tabulars . . . . .   | 52         |
| 18.3      | Dotted lines . . . . .  | 53         |
| 18.4      | Dotted lines which are no longer dotted . . . . .                                       | 54         |
| 18.5      | Dashed rules . . . . .  | 55         |
| 18.6      | Stacks of matrices . . . . .  | 55         |
| 18.7      | How to highlight cells of a matrix . . . . .  | 59         |
| 18.8      | Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code> . . . . .        | 61         |
| <b>19</b> | <b>Implementation</b>   | <b>61</b>  |
| <b>20</b> | <b>History</b>  | <b>260</b> |