

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

January 8, 2021

Abstract

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

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

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

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

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**.

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).

Important

Since the version 5.0 of `nicematrix`, one must use the letters `l`, `c` and `r` in the preambles of the environments and no longer the letters `L`, `C` and `R`.

For sake of compatibility with the previous versions, there exists an option `define-L-C-R` which must be used when loading `nicematrix`.

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

*This document corresponds to the version 5.9 of `nicematrix`, at the date of 2021/01/08.

1 The environments of this package

The package `nicematrix` defines the following new environments.

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

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

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

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

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

Important

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

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

2 The vertical space between the rows

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

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

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

New 5.9 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`.¹

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

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

¹One should remark that these parameters apply also to the columns of type `S` of `siunitx` whereas the package `cellspace` is not able to act on such columns of type `S`.

3 The vertical position of the arrays

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

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

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

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

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

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

1. an item

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

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

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

1. an item

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

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

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

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

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

4 The blocks

4.1 General case

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

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

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

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

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

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

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

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

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.²

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

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

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

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

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

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`).

²This 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.

```

\begin{NiceTabular}{cwc{2cm}wc{3cm}c}
rose      & tulipe & marguerite & dahlia \\
violette  &        &            &        \\
& \Block[draw=red,fill=red!15]{2-2}{\LARGE De très jolies fleurs} & & souci \\
pervenche & & & lys \\
arum      & iris & jacinthe & muguet
\end{NiceTabular}

```

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

New 5.7 As we can see on this example, it's possible to fill the block by using the key `fill` and to draw the frame with the key `draw`³. It's also possible to change the width (thickness) of that rules with the key `line-width`.

4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

- The natural width of the contents of these blocks is taken into account for the width of the current column.
- The specification of the horizontal position provided by the type of column (`c`, `r` or `l`) is taken into account for the blocks.
- The specifications of font specified for the column by a construction `>{...}` in the preamble of the array are taken into account for the mono-column blocks of that column (this behaviour is probably expected).

```

\begin{NiceTabular}{@{}>{\bfseries}lr@{}} \hline
\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 |
| | 18 |
| Sarah | 17 |
| | 15 |
| Ashley | 20 |
| Henry | 14 |
| | 15 |
| Madison | 19 |

4.3 The mono-row blocks

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

4.4 The mono-cell blocks

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

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

- It's possible to use the command `\` in a (mono-cell) block.

³If the key `draw` is used without value, the default color the rules of the tabulars is used

- 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`.⁴

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

| Write | 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.⁵

4.5 A small remark

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

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

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

5 The rules

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

⁴If one wishes to color the background of a unique celle, there is no point using the command `\Block`: it's possible to use the command `\cellcolor` (when the key `colortbl`-like is used).

⁵One may consider that the default value of the first mandatory argument of `\Block` is 1-1.

5.1 Some differences with the classical environments

5.1.1 The vertical rules

In the environments of `nicematrix`, the vertical rules specified by `|` in the preambles of the environments are never broken, even by an incomplete row or by a double horizontal rule specified by `\hline\hline` (there is no need to use `hhline`).⁶

```
\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.

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

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

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

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

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

However, in this case, it is probably more clever to add a command `\OnlyMainNiceMatrix` (cf. p. 33):

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

5.1.2 The command `\cline`

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

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

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

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

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

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

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

⁶This is the behaviour since the version 5.1 of `nicematrix`. Prior to that version, the behaviour was the standard behaviour of `array`.

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.

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

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

If one wishes to define new specifiers for columns in order to draw vertical rules (for example with a specific color or thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 33.

5.3 The keys `hlines` and `vlines`

The key `hlines` draws all the horizontal rules and the key `vlines` draws all the vertical rules excepted in the blocks (and the virtual blocks determined by dotted lines). In fact, in the environments with delimiters (as `{pNiceMatrix}` or `{bNiceArray}`) the exteriors rules are not drawn (as expected).

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

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

5.4 The key `hvlines`

The key `hvlines` draws all the vertical and horizontal rules (excepted in the blocks and the virtual blocks determined by dotted lines and excepted the rules corresponding of the frame of the blocks using the key `draw` which are drawn with their own characteristics).

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

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

5.5 The key `hvlines-except-corners`

The key `hvlines-except-corners` draws all the horizontal and vertical rules, excepted in the blocks (and the virtual blocks determined by dotted lines) and excepted in the empty corners.

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

As we can see, an “empty corner” is composed by the reunion of all the empty rectangles starting from the cell actually in the corner of the array.

It’s possible to give as value to the key `hvlines-except-corners` a list of the corners to take into consideration. The corners are designed by NW, SW, NE and SE (*north west*, *south west*, *north east* and *south east*).

```
\begin{NiceTabular}{*{6}{c}}%
[hvlines-except-corners=NE,cell-space-top-limit=3pt]
1\\
1&1\\
1&2&1\\
1&3&3&1\\
1&4&6&4&1\\
1&5&10&10&5&1
\end{NiceTabular}
```

| | | | | | |
|---|---|----|----|---|---|
| 1 | | | | | |
| 1 | 1 | | | | |
| 1 | 2 | 1 | | | |
| 1 | 3 | 3 | 1 | | |
| 1 | 4 | 6 | 4 | 1 | |
| 1 | 5 | 10 | 10 | 5 | 1 |

5.6 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.⁷

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

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

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

⁷The author of this document considers that type of construction as graphically poor.

5.7 Dotted rules

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

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

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

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

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

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

It's possible to change in `nicematrix` the letter used to specify a vertical dotted line with the option `letter-for-dotted-lines` available in `\NiceMatrixOptions`.

Remark : In the package `array` (on which the package `nicematrix` relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule⁸. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

6 The color of the rows and columns

6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

- The package `colortbl` patches `array`, leading to some incompatibilities (for example 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.

⁸In fact, this is true only for `\hline` and “|” but not for `\cline`: cf p. 7

6.2 The tools of nicematrix in the code-before

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.

The extension `nicematrix` provides a key `code-before` for some code that will be executed before the drawing of the tabular. In this `code-before`, new commands are available: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors` and `\chessboardcolors`.

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.

- 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 row and j the number of column of the cell.

```
\begin{NiceTabular}{|c|c|c|}[code-before =
\cellcolor{red!15}{3-1,2-2,1-3}]
\hline
a & b & c \\ \hline
e & f & g \\ \hline
h & i & j \\ \hline
\end{NiceTabular}
```

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

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

```
\begin{NiceTabular}{|c|c|c|}[code-before =
\rectanglecolor{blue!15}{2-2}{3-3}]
\hline
a & b & c \\ \hline
e & f & g \\ \hline
h & i & j \\ \hline
\end{NiceTabular}
```

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

- The command `\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).

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

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

- The command `\columncolor` takes its name from the command `\columncolor` of `colortbl`. Its syntax is similar to the syntax of `\rowcolor`.

- The command `\rowcolors` (with a *s*) takes its name from the command `\rowcolors` of `xcolor`⁹. 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.

New 5.8 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-j* 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`.

- **New 5.8** 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*.
- **New 5.8** With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.¹⁰
- 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`).

```
\begin{NiceTabular}{clr}
[hvlines,code-before = {\rowcolors{2}{blue!10}{}[cols=2-3,restart]]
\Block{1-}{Results} \\\
John & 12 \\\
Stephen & 8 \\\
Sarah & 18 \\\
Ashley & 20 \\\
Henry & 14 \\\
Madison & 15
\end{NiceTabular}
```

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

```
\begin{NiceTabular}{lrr}[hvlines,code-before =
\rowcolors{1}{blue!10}{}[respect-blocks]]
\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 command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

```
$$\begin{pNiceMatrix}[r,margin, code-before=\chessboardcolors{red!15}{blue!15}]
1 & -1 & 1 \\\
-1 & 1 & -1 \\\
1 & -1 & 1
\end{pNiceMatrix}$$
```

| | | |
|----|----|----|
| 1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | 1 |

⁹The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`.

¹⁰Otherwise, the color of a given row relies only upon the parity of its number.

We have used the key `r` which aligns all the columns rightwards (cf. p. 26).

One should remark that these commands are compatible with the commands of `booktabs` (`\toprule`, `\midrule`, `\bottomrule`, etc).

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

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

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

6.3 Color tools with the syntax of `colortbl`

It's possible to access the preceding tools with a syntax close to the syntax of `colortbl`. For that, one must use the key `colortbl-like` in the current environment.¹¹

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

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

```
\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 |

¹¹As of now, this key is not available in `\NiceMatrixOptions`.

7 The width of the columns

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

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

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

In the environments of `nicematrix`, it's also possible to fix the *minimal* width of all the columns of an array directly with the key `columns-width`.

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

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

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

It's possible to give the special value `auto` to the option `columns-width`: all the columns of the array will have a width equal to the widest cell of the array.¹²

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

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

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

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

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

But it's also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment `{NiceMatrixBlock}` with the option `auto-columns-width`¹³. 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}
```

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

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

¹²The 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).

¹³At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

Several compilations may be necessary to achieve the job.

8 The exterior rows and columns

The options `first-row`, `last-row`, `first-col` and `last-col` allow the composition of exterior rows and columns in the environments of `nicematrix`.

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

```

 $\begin{pNiceMatrix}[first-row,last-row,first-col,last-col]$ 
 $\begin{pNiceMatrix}[first-row,last-row,first-col,last-col,nullify-dots]$ 
& 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}
 $\end{pNiceMatrix}$ 

```

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

The dotted lines have been drawn with the tools presented p. 16.

We have several remarks to do.

- For the environments with an explicit preamble (i.e. `{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.
- 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. 28) is used, `nicematrix` has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (but not the number of columns).
 - In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

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

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

```

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

```

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

Remarks

- As shown in the previous example, the horizontal and vertical rules doesn't extend in the exterior rows and columns.
However, if one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 33.
- A specification of color present in `code-for-first-row` also applies to a dotted line draw in this exterior “first row” (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 14) 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).

9 The continuous dotted lines

Inside the environments of the package `nicematrix`, new commands are defined: `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots`. These commands are intended to be used in place of `\dots`, `\cdots`, `\vdots`, `\ddots` and `\iddots`.¹⁴

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells¹⁵ 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

¹⁴The 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`.

¹⁵The precise definition of a “non-empty cell” is given below (cf. p. 34).

to change the color of these lines with the option `color`.¹⁶

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

$$\begin{bmatrix} a_1 & \cdots & & & a_1 \\ \vdots & a_2 & \cdots & & a_2 \\ & \vdots & \ddots[\textcolor{red}] & & \\ \\ a_1 & a_2 & & & a_n \end{bmatrix}$$

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

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

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

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

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

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

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

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

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\Vdots` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.¹⁷

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

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

$$\begin{bmatrix} 0 & \cdots & \hspace{1cm} & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & & 0 \\ \vdots & & & \vdots \end{bmatrix}$$

¹⁶It's also possible to change the color of all theses dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 20.

¹⁷In `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. 14.

9.1 The option nullify-dots

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

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

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

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

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

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

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

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

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

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).

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

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

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

9.2 The commands `\Hdotsfor` and `\Vdotsfor`

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

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

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

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

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

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

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used when the package `colortbl` 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
& \Vdotsfor{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} & \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}
```

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

9.3 How to generate the continuous dotted lines transparently

Imagine you have a document with a great number of mathematical matrices with ellipsis. You may wish to use the dotted lines of `nicematrix` without having to modify the code of each matrix. It's possible with the keys. `renew-dots` and `renew-matrix`.¹⁸

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`¹⁴ and `\hdotsfor` are redefined within the environments provided by `nicematrix` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` (“automatic dots” of `amsmath`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

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

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

¹⁸The options `renew-dots`, `renew-matrix` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage`. There is also a key `transparent` which is an alias for the conjunction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.

nicematrix.

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

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

9.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `code-after` which is described p. 21) accept two optional arguments specified by the tokens `_` and `^` for labels positionned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

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

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

9.5 Customization of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and by the command `\line` in the `code-after` which is described p. 21) may be customized by three options (specified between square brackets after the command):

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

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

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

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

The option `xdots/color`

The option `xdots/color` fixes the color of the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 15.

The option `xdots/shorten`

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

The option `xdots/line-style`

It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).¹⁹

¹⁹The 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.

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

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

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

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

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

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

9.6 The dotted lines and the rules

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

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

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

10 The code-after

The option `code-after` may be used to give some code that will be executed after the construction of the matrix.²⁰

A special command, called `\line`, is available to draw directly dotted lines between nodes. It takes two arguments for the two cells to rely, both of the form `i-j` where `i` is the number of row and `j` is the number of column. It may be used, for example, to draw a dotted line between two adjacent cells.

```
\NiceMatrixOptions{xdots/shorten = 0.6 em}
\begin{pNiceMatrix}[code-after=\line{2-2}{3-3}]
I      & 0      & \Cdots & 0      & \\
0      & I      & \Ddots & \Vdots & \\
\Vdots & \Ddots & I      & 0      & \\
0      & \Cdots & 0      & I      & \\
\end{pNiceMatrix}
```

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

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` (for an example, cf. p. 39).

²⁰There is also a key `code-before` described p. 11.

11 The notes in the tabulars

11.1 The footnotes

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

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

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

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

11.2 The notes of tabular

The package `nicematrix` also provides a command `\tabularnote` which gives the ability to specify notes that will be composed at the end of the array with a width of line equal to the width of the array (excepted the potential exterior columns). With no surprise, that command is available only in the environments without delimiters, that is to say `{NiceTabular}`, `{NiceArray}` and `{NiceMatrix}`.

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

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

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

^a Achard is an old family of the Poitou.

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

- If you have several successive commands `\tabularnote{...}` with no space at all between them, the labels of the corresponding notes are composed together, separated by commas (this is similar to the option `multiple` of `footmisc` for the footnotes).

Table 1: Use of `\tabularnote`^a

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

Some text before the notes.

^a It's possible to put a note in the caption.
^b Considered as the first nurse of history.
^c Nicknamed "the Lady with the Lamp".
^d The label of the note is overlapping.

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

For an illustration of some of those remarks, see table 1, p. 23. This table has been composed with the following code.

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

11.3 Customisation of the tabular notes

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

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

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

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

We detail these keys.

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

Initial value: `false`

That key is also available within a given environment.

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

Initial value: `false`

That key is also available within a given environment.

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

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

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

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

Initial value: `#1`

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

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

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

Initial value: `#1`

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

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

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

- The notes are composed at the end of the tabular by using internally a style of list of `enumitem`. The key `notes/enumitem-keys` specifies a list of pairs `key=value` (following the specifications of `enumitem`) to customize that type of list.

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

This initial value contains the specification `align = left` which requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 23).

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

Initial value: `afterlabel = \nobreak, itemjoin = \quad`

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

Initial value: `empty`

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

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

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

For an example of customization of the tabular notes, see p. 35.

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

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

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

12 Other features

12.1 Use of the column type S of siunitx

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

```


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


```

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

12.2 Alignment option in `{NiceMatrix}`

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

```


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


```

12.3 The command `\rotate`

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

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

```


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


```

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

```


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


```

12.4 The option `small`

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

```

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

```

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

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

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

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

12.5 The counters `iRow` and `jCol`

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column²¹. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `code-before` (cf. p. 11) and in the `code-after` (cf. p. 21), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

```

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

²¹We 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.

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, excepted in the cells of the potential exterior rows and columns).

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

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

12.6 The option light-syntax

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

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

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

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

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

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.²²

12.7 Color of the delimiters

For the environements with delimiters (`\pNiceArray`, `\pNiceMatrix`, etc.), it's possible to change the color of the delimiters with the key `delimiters-color`.

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

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

12.8 The environment \NiceArrayWithDelims

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

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

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

²²The reason is that, when the option `light-syntax` is used, the whole content of the environment is loaded as a TeX argument to be analyzed. The environment doesn't behave in that case as a standard environment of LaTeX which only put TeX commands before and after the content.

13 Use of Tikz with nicematrix

13.1 The nodes corresponding to the contents of the cells

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

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

However, it's possible to impose the creation of a node with the command `\NotEmpty`. ²³

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

In the environment with the number n , the node of the row i and column j has for name `nm-n-i-j`.

The command `\NiceMatrixLastEnv` provides the number of the last environment of `nicematrix` (for LaTeX, it's a “fully expandable” command and not a counter).

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

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

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

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

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

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

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

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 39).

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

²³One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 16).

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

In fact, the package `nicematrix` can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option `create-medium-nodes` and the second ones with the option `create-large-nodes`.²⁴

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

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

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

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.²⁵

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

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.²⁶

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

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

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

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

Here is an array composed with the following code:

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

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

²⁴There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

²⁵There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 15).

²⁶The 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, we have colored all the cells of the array with `\chessboardcolors`.

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

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

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

13.3 The “row-nodes” and the “col-nodes”

The package `nicematrix` creates a PGF/Tikz node indicating the potential position of each horizontal rule (with the names `row-i`) and each vertical rule (with the names `col-j`), as described in the following figure. These nodes are available in the `code-before` and the `code-after`.

| | | | | |
|-------|--------|--------|----------|-------|
| row-1 | rose | tulipe | lys | |
| row-2 | arum | iris | violette | |
| row-3 | muguet | dahlia | souci | |
| row-4 | col-1 | col-2 | col-3 | col-4 |

If we use Tikz (we remind that `nicematrix` does not load Tikz by default), we can access (in the `code-before` and the `code-after`) to the intersection of the horizontal rule *i* and the vertical rule *j* with the syntax `(row-i-|col-j)`.

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

| | | | | | | | | |
|---|---|----|----|----|----|----|---|---|
| 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 |

14 API for the developpers

The package `nicematrix` provides two variables which are internal but public²⁷:

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

These variables contain the code of what we have called the “code-before” and the “code-after”. The developer can use them to add code from a cell of the array (the affectation must be global, allowing to exit the cell, which is a TeX group).

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

Example : We want to write a command `\hatchcell` to hatch the current cell (with an optional argument between brackets for the color). It's possible to program such command `\hatchcell` as follows, explicitly using the public variable `\g_nicematrix_code_before_tl` (this code requires the Tikz library `patterns`: `\usetikzlibrary{patterns}`).

```
ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_hatchcell:nnn
{
  \begin { tikzpicture }
  \fill [ pattern = north~west~lines , pattern~color = #3 ]
    ( row - #1 -| col - #2 )
    rectangle
    ( row - \int_eval:n { #1 + 1 } -| col - \int_eval:n { #2 + 1 } ) ;
  \end { tikzpicture }
}

\NewDocumentCommand \hatchcell { ! O { black } }
{
  \tl_gput_right:Nx \g_nicematrix_code_before_tl
  {
    \__pantigny_hatchcell:nnn
    { \int_use:c { c@iRow } }
    { \int_use:c { c@jCol } }
    { #1 }
  }
}
\ExplSyntaxOff
```

Here is an example of use:

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

| | | |
|-------------|--------|--------|
| Tokyo | Paris | London |
| Lima | Oslo | Miami |
| Los Angeles | Madrid | Roma |

²⁷According to the LaTeX3 conventions, each variable with name beginning with `\g_nicematrix` ou `\l_nicematrix` is public and each variable with name beginning with `\g__nicematrix` or `\l__nicematrix` is private.

15 Technical remarks

15.1 Definition of new column types

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

For example, one may wish to define a new column type `?` in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job²⁸:

```
\newcolumnstype{?}{!\OnlyMainNiceMatrix{\vrule width 1 pt}}
```

The heavy vertical rule won't extend in the exterior rows.²⁹

```
\begin{pNiceArray}{cc?cc}[first-row,last-row=3]
C_1 & C_2 & C_3 & C_4 \\
a & b & c & d \\
e & f & g & h \\
C_1 & C_2 & C_3 & C_4 \\
\end{pNiceArray}
```

$$\begin{pmatrix} C_1 & C_2 & C_3 & C_4 \\ a & b & c & d \\ e & f & g & h \\ C_1 & C_2 & C_3 & C_4 \end{pmatrix}$$

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

15.2 Diagonal lines

By default, all the diagonal lines³⁰ of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That's why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

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

Example with parallelization (default):

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

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

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

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

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

The same example without parallelization:

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

²⁸The command `\vrule` is a TeX (and not LaTeX) command.

²⁹Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

³⁰We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

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

15.3 The “empty” cells

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

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

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

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

- Each cell whose TeX output has a width equal to zero is empty.
- A cell containing the command `\NotEmpty` is not empty (and a PGF/Tikz node) is created in that cell.
- A cell with a command `\Hspace` (or `\Hspace*`) is empty. This command `\Hspace` is a command defined by the package `nicematrix` with the same meaning as `\hspace` except that the cell where it is used is considered as empty. This command can be used to fix the width of some columns of the matrix without interfering with `nicematrix`.

15.4 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea³¹. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`³². The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

15.5 Incompatibilities

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

³¹In 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).*

³²And not by inserting `@{}` on both sides of the preamble of the array. As a consequence, the length of the `\hline` is not modified and may appear too long, in particular when using square brackets

16 Examples

16.1 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 11 p. 22.

Let's consider that we wish to number the notes of a tabular with stars.³³

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 ³⁴

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

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

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

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

³³Of course, it's realistic only when there is very few notes in the tabular.

³⁴In 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.

16.2 Dotted lines

A permutation matrix (as an example, we have raised the value of `xdots/shorten`).

```

 $\begin{pNiceMatrix}[xdots/shorten=0.6em]$ 
0      & 1 & 0 & & & \Cdots & 0 & \\
\Vdots & & & & \Ddots & & & \Vdots \\
      & & & & \Ddots & & & \\
      & & & & \Ddots & & 0 & \\
0      & 0 & & & & & 1 & \\
1      & 0 & & & \Cdots & & 0 & \\
 $\end{pNiceMatrix}$ 

```

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

An example with `\Iddots` (we have raised again the value of `xdots/shorten`).

```

 $\begin{pNiceMatrix}[xdots/shorten=0.9em]$ 
1      & \Cdots & & 1 & \\
\Vdots & & & 0 & \\
      & \Iddots & \Iddots & \Vdots & \\
1      & 0 & & \Cdots & 0 \\
 $\end{pNiceMatrix}$ 

```

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

An example with `\multicolumn`:

```

 $\begin{BNiceMatrix}[nullify-dots]$ 
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\Cdots & & \multicolumn{6}{C}{10 \text{ other rows}} & & \Cdots \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 $\end{BNiceMatrix}$ 

```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix}$$

An example with `\Hdotsfor`:

```
\begin{pNiceMatrix}[nullify-dots]
0 & 1 & 1 & 1 & 1 & 0 & \\
0 & 1 & 1 & 1 & 1 & 0 & \\
\Vdots & & \Hdotsfor{4} & & \Vdots & & \\
& & \Hdotsfor{4} & & & & \\
& & \Hdotsfor{4} & & & & \\
& & \Hdotsfor{4} & & & & \\
0 & 1 & 1 & 1 & 1 & 0 & \\
\end{pNiceMatrix}
```

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ \vdots & \dots & \dots & \dots & \dots & \vdots \\ & \dots & \dots & \dots & \dots & \\ & \dots & \dots & \dots & \dots & \\ & \dots & \dots & \dots & \dots & \\ 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

An example for the resultant of two polynoms:

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

$$\left| \begin{array}{ccccccc} a_0 & & & & b_0 & & \\ a_1 & & \dots & & b_1 & & \dots \\ \vdots & & \dots & & \vdots & & \dots \\ a_p & & & a_0 & & & b_1 \\ & & \dots & & a_1 & & \vdots \\ & & & \vdots & & & \vdots \\ & & & a_p & & & b_q \end{array} \right|$$

An example for a linear system:

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

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

16.3 Dotted lines which are no longer dotted

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

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

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

16.4 Width of the columns

In the following example, we use `{NiceMatrixBlock}` with the option `auto-columns-width` because we want the same automatic width for all the columns of the matrices.

```
\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions
  { last-col,code-for-last-col = \color{blue}\scriptstyle,light-syntax}
\setlength{\extrarowheight}{1mm}
$\begin{pNiceArray}{cccc:c}
  1 1 1 1 1 {} ;
  2 4 8 16 9 ;
  3 9 27 81 36 ;
  4 16 64 256 100
\end{pNiceArray}$
\medskip
$\begin{pNiceArray}{cccc:c}
  1 1 1 1 1 ;
  0 2 6 14 7 { L_2 \gets -2 L_1 + L_2 } ;
  0 6 24 78 33 { L_3 \gets -3 L_1 + L_3 } ;
  0 12 60 252 96 { L_4 \gets -4 L_1 + L_4 }
\end{pNiceArray}$
...
\end{NiceMatrixBlock}
```

$$\begin{array}{c}
\begin{pmatrix} 1 & 1 & 1 & 1 & \vdots & 1 \\ 2 & 4 & 8 & 16 & \vdots & 9 \\ 3 & 9 & 27 & 81 & \vdots & 36 \\ 4 & 16 & 64 & 256 & \vdots & 100 \end{pmatrix} \\
\begin{pmatrix} 1 & 1 & 1 & 1 & \vdots & 1 \\ 0 & 2 & 6 & 14 & \vdots & 7 \\ 0 & 6 & 24 & 78 & \vdots & 33 \\ 0 & 12 & 60 & 252 & \vdots & 96 \end{pmatrix} \begin{array}{l} L_2 \leftarrow -2L_1 + L_2 \\ L_3 \leftarrow -3L_1 + L_3 \\ L_4 \leftarrow -4L_1 + L_4 \end{array} \\
\begin{pmatrix} 1 & 1 & 1 & 1 & \vdots & 1 \\ 0 & 1 & 3 & 7 & \vdots & \frac{7}{2} \\ 0 & 3 & 12 & 39 & \vdots & \frac{33}{2} \\ 0 & 1 & 5 & 21 & \vdots & 8 \end{pmatrix} \begin{array}{l} L_2 \leftarrow \frac{1}{2}L_2 \\ L_3 \leftarrow \frac{1}{2}L_3 \\ L_4 \leftarrow \frac{1}{12}L_4 \end{array}
\end{array}
\quad
\begin{array}{c}
\begin{pmatrix} 1 & 1 & 1 & 1 & \vdots & 1 \\ 0 & 1 & 3 & 7 & \vdots & \frac{7}{2} \\ 0 & 0 & 3 & 18 & \vdots & 6 \\ 0 & 0 & -2 & -14 & \vdots & -\frac{9}{2} \end{pmatrix} \begin{array}{l} L_3 \leftarrow -3L_2 + L_3 \\ L_4 \leftarrow L_2 - L_4 \end{array} \\
\begin{pmatrix} 1 & 1 & 1 & 1 & \vdots & 1 \\ 0 & 1 & 3 & 7 & \vdots & \frac{7}{2} \\ 0 & 0 & 1 & 6 & \vdots & 2 \\ 0 & 0 & -2 & -14 & \vdots & -\frac{9}{2} \end{pmatrix} \begin{array}{l} L_3 \leftarrow \frac{1}{3}L_3 \\ L_4 \leftarrow -L_3 + L_4 \end{array} \\
\begin{pmatrix} 1 & 1 & 1 & 1 & \vdots & 1 \\ 0 & 1 & 3 & 7 & \vdots & \frac{7}{2} \\ 0 & 0 & 1 & 6 & \vdots & 2 \\ 0 & 0 & 0 & -2 & \vdots & -\frac{1}{2} \end{pmatrix} \begin{array}{l} L_4 \leftarrow L_3 + L_4 \end{array}
\end{array}$$

16.5 How to highlight cells of a matrix

In order to highlight a cell of a matrix, it's possible to “draw” that cell with the key `draw` of the command `\Block` (this is one of the uses of a mono-cell block³⁵).

```

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

```

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

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 command `\hline`, the specifier “|” and the options `hlines`, `vlines` and `hvlines` spread the cells.³⁶

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). However, it's not possible to do a fine tuning. That's why we describe now a method to highlight a row of the matrix. We create a rectangular Tikz node which encompasses the nodes of the second row with the Tikz library `fit`. This Tikz node is filled after the construction of the matrix. In order to see the text *under* this node, we have to use transparency with the `blend` mode equal to `multiply`.

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

```

\usepackage{tikz}
\usetikzlibrary{fit}

```

³⁵We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

³⁶For the command `\cline`, see the remark p. 7.

```

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

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

```

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```



```

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

```

We obtain the following matrix.

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

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

```

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

```

We obtain the following matrix.

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

In the following example, we use the “large nodes” to highlight a zone of the matrix.

```

\begin{pNiceArray}{>{\strut}cccc}[create-large-nodes,margin,extra-margin=2pt]
  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} \\
\CodeAfter
  \tikz \path [name suffix = -large,fill = red!15, blend mode = multiply]
    (1-1.north west)
    |- (2-2.north west)
    |- (3-3.north west)
    |- (4-4.north west)
    |- (4-4.south east)
    |- (1-1.north west) ;
\end{pNiceArray}

```

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

16.6 Direct use of the Tikz nodes

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

The use of `\NiceMatrixBlock` with the option `auto-columns-width` gives the same width for all the columns and, therefore, a perfect alignment of the two superposed matrices.

```

\begin{NiceMatrixBlock}[auto-columns-width]

```

```
\NiceMatrixOptions{nullify-dots}
```

The three matrices will be displayed using an environment `{array}` (an environment `{tabular}` may also be possible).

```

 $\begin{array}{cc}$ 
 $\&$ 

```

The matrix B has a “first row” (for C_j) and that’s why we use the key `first-row`.

```

\begin{bNiceArray}{c>\strut}cccc[name=B,first-row]
      & & & C_j & \\
b_{11} & \& \cdots & \& b_{1j} & \& \cdots & \& b_{1n} \\
\vdots & & & \vdots & & & \vdots & \\
      & & & b_{kj} & & & \\
      & & & \vdots & & & \\
b_{n1} & \& \cdots & \& b_{nj} & \& \cdots & \& b_{nn} \\
\end{bNiceArray} \& \&

```

The matrix A has a “first column” (for L_i) and that’s why we use the key `first-col`.

```

\begin{bNiceArray}{cc>\strut}ccc[name=A,first-col]
      & a_{11} & \& \cdots & \& & & a_{1n} \\
      & \vdots & & & & & \vdots & \\
L_i & a_{i1} & \& \cdots & \& a_{ik} & \& \cdots & \& a_{in} \\
      & \vdots & & & & & \vdots & \\
      & a_{n1} & \& \cdots & \& & & a_{nn} \\
\end{bNiceArray}
\&

```

In the matrix product, the two dotted lines have an open extremity.

```

\begin{bNiceArray}{cc>\strut}ccc
      & \& & & \& \& \\
      & \& \vdots & & \& \& \\
\cdots & \& c_{ij} & & \& \& \\
\& \\
\& \\
\end{bNiceArray}
\end{array}$

```

```
\end{NiceMatrixBlock}
```

```

\begin{tikzpicture}[remember picture, overlay]
\node [highlight = (A-3-1) (A-3-5) ] {} ;
\node [highlight = (B-1-3) (B-5-3) ] {} ;
\draw [color = gray] (A-3-3) to [bend left] (B-3-3) ;
\end{tikzpicture}

```

$$L_i \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{i1} & \cdots & a_{in} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} \vdots \\ \vdots \\ c_{ij} \\ \vdots \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & \cdots & b_{1j} & \cdots & b_{1n} \\ \vdots & & \vdots & & \vdots \\ b_{n1} & \cdots & b_{nj} & \cdots & b_{nn} \end{bmatrix}$$

17 Implementation

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

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

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

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

Declaration of the package and packages loaded

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

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

`<@@=nicematrix>`

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

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

We give the traditional declaration of a package written with `expl3`:

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

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

We load some packages.

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

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

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

Technical definitions

```

21 \bool_new:N \c_@@_in_preamble_bool
22 \bool_set_true:N \c_@@_in_preamble_bool
23 \AtBeginDocument { \bool_set_false:N \c_@@_in_preamble_bool }

24 \bool_new:N \c_@@_booktabs_loaded_bool
25 \bool_new:N \c_@@_enumitem_loaded_bool
26 \bool_new:N \c_@@_tikz_loaded_bool
27 \AtBeginDocument
28 {
29   \ifpackageloaded { booktabs }
30     { \bool_set_true:N \c_@@_booktabs_loaded_bool }
31     { }
32   \ifpackageloaded { enumitem }
33     { \bool_set_true:N \c_@@_enumitem_loaded_bool }
34     { }
35   \ifpackageloaded { tikz }
36     {

```

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.

```

37   \bool_set_true:N \c_@@_tikz_loaded_bool
38   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
39   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
40 }
41 {
42   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
43   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
44 }
45 }

```

We test whether the current class is `revtex4-1` or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming.

```

46 \bool_new:N \c_@@_revtex_bool
47 \ifclassloaded { revtex4-1 }
48 { \bool_set_true:N \c_@@_revtex_bool }
49 { }
50 \ifclassloaded { revtex4-2 }
51 { \bool_set_true:N \c_@@_revtex_bool }
52 { }

```

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.

```

53 \ProvideDocumentCommand \iddots { }
54 {
55   \mathinner
56   {
57     \tex_mkern:D 1 mu
58     \box_move_up:nn { 1 pt } { \hbox:n { . } }
59     \tex_mkern:D 2 mu
60     \box_move_up:nn { 4 pt } { \hbox:n { . } }
61     \tex_mkern:D 2 mu
62     \box_move_up:nn { 7 pt }
63     { \vbox:n { \kern 7 pt \hbox:n { . } } }
64     \tex_mkern:D 1 mu
65   }
66 }

```

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.

```

67 \AtBeginDocument
68 {
69   \@ifpackageloaded { booktabs }
70   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
71   { }
72 }
73 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
74 {
75   \cs_set_eq:NN \@_old_pgful@check@rerun \pgfutil@check@rerun

```

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

```

76   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
77   {
78     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
79     { \@_old_pgful@check@rerun { ##1 } { ##2 } }
80   }
81 }

```

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

```

82 \bool_new:N \c_@@_colortbl_loaded_bool
83 \AtBeginDocument
84 {
85   \@ifpackageloaded { colortbl }
86   { \bool_set_true:N \c_@@_colortbl_loaded_bool }
87   {

```

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. Idem for

```

88   \cs_set_protected:Npn \CT@arc@ { }
89   \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
90   \cs_set:Npn \CT@arc@ #1 #2
91   {
92     \dim_compare:nNt \baselineskip = \c_zero_dim \noalign
93     { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
94   }

```

Idem for `\CT@drs@`.

```

95   \cs_set_protected:Npn \CT@drsc@ { }
96   \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
97   \cs_set:Npn \CT@drs@ #1 #2
98   {
99     \dim_compare:nNt \baselineskip = \c_zero_dim \noalign
100    { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
101  }
102  \cs_set:Npn \hline
103  {
104    \noalign { \ifnum 0 = ` } \fi
105    \cs_set_eq:NN \hskip \vskip
106    \cs_set_eq:NN \vrule \hrule
107    \cs_set_eq:NN \@width \@height
108    { \CT@arc@ \vline }
109    \futurelet \reserved@a
110    \@xhline
111  }
112 }
113 }

```

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.

```

114 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
115 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
116 {
117   \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
118   \int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
119   \multispan { \int_eval:n { #2 - #1 + 1 } }
120   {
121     \CT@arc@
122     \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`³⁷

```

123   \skip_horizontal:N \c_zero_dim
124 }

```

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`.

```

125   \everycr { }
126   \cr
127   \noalign { \skip_vertical:N -\arrayrulewidth }
128 }

```

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.

```

129 \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`.

```

130 { \@@_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*.

```

131 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1-#2 \q_stop }
132 \cs_set:Npn \@@_cline_i:w #1-#2-#3 \q_stop
133 {

```

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).

```

134   \int_compare:nNnT { #1 } < { #2 }
135   { \multispan { \int_eval:n { #2 - #1 } } & }
136   \multispan { \int_eval:n { #3 - #2 + 1 } }
137   {
138     \CT@arc@
139     \leaders \hrule \@height \arrayrulewidth \hfill
140     \skip_horizontal:N \c_zero_dim
141   }

```

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

```

142   \peek_meaning_remove_ignore_spaces:NTF \cline
143   { & \@@_cline_i:en { \@@_succ:n { #3 } } }
144   { \everycr { } \cr }
145 }
146 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

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

```

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

```

The following command is a small shortcut.

```

149 \cs_new:Npn \@@_math_toggle_token:

```

³⁷See question 99041 on TeX StackExchange.

```

150 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

151 \cs_new_protected:Npn \@@_set_CT@arc@:
152 { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
153 \cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
154 { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
155 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
156 { \cs_set:Npn \CT@arc@ { \color { #1 } } }

```

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.

```

157 \bool_new:N \c_@@_siunitx_loaded_bool
158 \AtBeginDocument
159 {
160   \@ifpackageloaded { siunitx }
161     { \bool_set_true:N \c_@@_siunitx_loaded_bool }
162     { }
163 }

```

The command `\NC@rewrite@S` is a LaTeX command created by siunitx in connection with the S column. In the code of siunitx, this command is defined by:

```

\renewcommand*{\NC@rewrite@S}[1] []
{
  \@temptokena \exp_after:wN
  {
    \tex_the:D \@temptokena
    > { \__siunitx_table_collect_begin: S {#1} }
    c
    < { \__siunitx_table_print: }
  }
  \NC@find
}

```

We want to patch this command (in the environments of nicematrix) in order to have:

```

\renewcommand*{\NC@rewrite@S}[1] []
{
  \@temptokena \exp_after:wN
  {
    \tex_the:D \@temptokena
    > { \@@_Cell: \__siunitx_table_collect_begin: S {#1} }
    \@@_true_c:
    < { \__siunitx_table_print: \@@_end_Cell: }
  }
  \NC@find
}

```

However, we don't want to use explicitly any private command of siunitx. That's why we will extract the name of the two `__siunitx...` commands by their position in the code of `\NC@rewrite@S`.

Since the command `\NC@rewrite@S` appends some tokens to the *toks* list `\@temptokena`, we use the LaTeX command `\NC@rewrite@S` in a group (`\group_begin:-\group_end:`) and we extract the two command names which are in the *toks* `\@temptokena`. However, this extraction can be done only when siunitx is loaded (and it may be loaded after nicematrix) and, in fact, after the beginning of the document — because some instructions of siunitx are executed in a `\AtBeginDocument`). That's why this extraction will be done only at the first use of an environment of nicematrix with the command `\@@_adapt_S_column:`.

```

164 \cs_set_protected:Npn \@@_adapt_S_column:
165 {
166   \bool_if:NT \c_@@_siunitx_loaded_bool
167   {
168     \group_begin:
169     \@temptokena = { }

```

We protect `\NC@find` which is at the end of `\NC@rewrite@S`.

```
170 \cs_set_eq:NN \NC@find \prg_do_nothing:
171 \NC@rewrite@S { }
```

Conversion of the *toks* `\@temptokena` in a token list of `expl3` (the *toks* are not supported by `expl3` but we can, nevertheless, use the option `V` for `\tl_gset:NV`).

```
172 \tl_gset:NV \g_tmpa_tl \@temptokena
173 \group_end:
174 \tl_new:N \c_@@_table_collect_begin_tl
175 \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
176 \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
177 \tl_new:N \c_@@_table_print_tl
178 \tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl { -1 } }
```

The token lists `\c_@@_table_collect_begin_tl` and `\c_@@_table_print_tl` contain now the two commands of `siunitx`.

If the adaptation has been done, the command `\@@_adapt_S_column:` becomes no-op (globally).

```
179 \cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
180 }
181 }
```

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

```
182 \AtBeginDocument
183 {
184 \bool_if:nTF { ! \c_@@_siunitx_loaded_bool }
185 { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
186 {
187 \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
188 {
189 \renewcommand*{\NC@rewrite@S}[1] []
190 {
191 \@temptokena \exp_after:wN
192 {
193 \tex_the:D \@temptokena
194 > { \c_@@_table_collect_begin_tl S {##1} }

```

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

```
195 \@@_true_c:
196 < { \c_@@_table_print_tl \@@_end_Cell: }
197 }
198 \NC@find
199 }
200 }
201 }
202 }
```

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

```
203 \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 avoid that situation.

```
204 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
205 {
206 \iow_now:Nn \@mainaux
207 {
208 \ExplSyntaxOn
209 \cs_if_free:NT \pgfsyspdfmark
210 { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
```



```

211     \ExplSyntaxOff
212   }
213   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
214 }

```

Parameters

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

```

215 \bool_new:N \c_@@_define_L_C_R_bool
216 \cs_new_protected:Npn \@@_define_L_C_R:
217 {
218   \newcolumntype L l
219   \newcolumntype C c
220   \newcolumntype R r
221 }

```

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.

```

222 \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).

```

223 \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.

```

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

```

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

```

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

```

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

```

228 \int_new:N \g_@@_NiceMatrixBlock_int

```

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

```

229 \dim_new:N \l_@@_columns_width_dim

```

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

```

230 \tl_new:N \l_@@_cell_type_tl
231 \tl_set:Nn \l_@@_cell_type_tl { c }

```

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

```

232 \dim_new:N \g_@@_blocks_wd_dim

```

Idem pour the blocks mono-row.

```
233 \dim_new:N \g_@@_blocks_ht_dim
234 \dim_new:N \g_@@_blocks_dp_dim
```

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

```
235 \seq_new:N \g_@@_names_seq
```

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

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

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

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

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

```
238 \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.

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

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

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

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

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

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

The following colors will be used to memorize the color of the potential “first col” and the potential “first row”.

```
248 \colorlet { nicematrix-last-col } { . }
249 \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`).

```
250 \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*.

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

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

```

252 \cs_new:Npn \@@_full_name_env:
253 {
254   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
255   { command \space \c_backslash_str \g_@@_name_env_str }
256   { environment \space \{ \g_@@_name_env_str \} }
257 }
```

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`).

```

258 \tl_new:N \g_nicematrix_code_after_tl
```

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

```

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

The counters `\l_@@_old_iRow_int` and `\l_@@_old_jCol_int` will be used to save the values of the potential LaTeX counters `iRow` and `jCol`. These LaTeX counters will be restored at the end of the environment.

```

260 \int_new:N \l_@@_old_iRow_int
261 \int_new:N \l_@@_old_jCol_int
```

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

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

```

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

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

```

263 \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).

```

264 \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).

```

265 \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`.

```

266 \tl_new:N \l_@@_code_before_tl
267 \bool_new:N \l_@@_code_before_bool

```

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

```

268 \dim_new:N \l_@@_x_initial_dim
269 \dim_new:N \l_@@_y_initial_dim
270 \dim_new:N \l_@@_x_final_dim
271 \dim_new:N \l_@@_y_final_dim

```

expl3 provides scratch dimension `\l_tmpa_dim` and `\l_tmpd_dim`. We create two other in the same spirit (if they don't exist yet : that's why we use `\dim_zero_new:N`).

```

272 \dim_zero_new:N \l_tmpc_dim
273 \dim_zero_new:N \l_tmpd_dim

```

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

```

274 \bool_new:N \g_@@_empty_cell_bool

```

The following dimension will be used to save the current value of `\arraycolsep`.

```

275 \dim_new:N \@@_old_arraycolsep_dim

```

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

```

276 \dim_new:N \g_@@_width_last_col_dim
277 \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 braces:

`{imin}{jmin}{imax}{jmax}{options}{contents}`.

The variable is global because it will be modified in the cells of the array.

```

278 \seq_new:N \g_@@_blocks_seq

```

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

```

279 \seq_new:N \g_@@_pos_of_blocks_seq

```

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

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

```

280 \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 `color=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.

```

281 \seq_new:N \g_@@_pos_of_stroken_blocks_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).

```

282 \int_new:N \g_@@_static_num_of_col_int

```

The following token lists correspond to the keys `fill` and `draw` of a command `\Block`.

```
283 \tl_new:N \l_@@_fill_tl
284 \tl_new:N \l_@@_draw_tl
```

The following token list correspond to the key `color` of the command `\Block`. However, as of now (v. 5.7 of `nicematrix`), the key `color` linked to `fill` with an error. We will give to the key `color` of `\Block` its new meaning in a few months (with its new definition, the key `color` will draw the frame with the given color but also color the content of the block (that is to say the text) as does the key `color` of a Tikz node).

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

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

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

The parameter of position of the label of a block (`c`, `r` or `l`).

```
287 \tl_new:N \l_@@_pos_of_block_tl
288 \tl_set:Nn \l_@@_pos_of_block_tl { c }
```

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

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

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

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

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.

```
291 \int_new:N \l_@@_first_row_int
292 \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.

```
293 \int_new:N \l_@@_first_col_int
294 \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).

```
295 \int_new:N \l_@@_last_row_int
296 \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”.³⁸

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

Idem for `\l_@@_last_col_without_value_bool`

```
298 \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.

```
299 \int_new:N \l_@@_last_col_int
300 \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.

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

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

The command `\tablarnote`

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

```
302 \newcounter { tablarnote }
```

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

```
303 \seq_new:N \g_@@_tablarnotes_seq
```

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

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

³⁸We 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 counter will be used to count the number of successive tabular notes such as in `\tabularnote{Note 1}\tabularnote{Note 2}\tabularnote{Note 3}`. In the tabular, the labels of those nodes are composed as a comma separated list (e.g. a,b,c).

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

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

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

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

```
307 \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`.

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

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

```
309 \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`).

```
310 \AtBeginDocument
311 {
312   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
313   {
314     \NewDocumentCommand \tabularnote { m }
315     { \@@_error:n { enumitem-not-loaded } }
316   }
317 }
```

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.

```
318   \newlist { tabularnotes } { enumerate } { 1 }
319   \setlist [ tabularnotes ]
320   {
321     topsep = 0pt ,
322     noitemsep ,
323     leftmargin = * ,
324     align = left ,
325     labelsep = 0pt ,
326     label =
327       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
328   }
329   \newlist { tabularnotes* } { enumerate* } { 1 }
330   \setlist [ tabularnotes* ]
331   {
332     afterlabel = \nobreak ,
333     itemjoin = \quad ,
334     label =
335       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
336   }
```

The command `\tabularnote` is available in the whole document (and not only in the environments of `nicematrix`) because we want it to be available in the caption of a `{table}` (before the following `{NiceTabular}` or `{NiceArray}`). That's also the reason why the variables `\c@tabularnote` and `\g_@@_tabularnotes_seq` will be cleared at the end of the environment of `nicematrix` (and not at the beginning).

Unfortunately, if the package `caption` is loaded, the command `\caption` evaluates its argument twice and since it is not aware (of course) of `\tabularnote`, the command `\tabularnote` is, in fact, not usable in `\caption` when `caption` is loaded.³⁹

```

337     \NewDocumentCommand \tabularnote { m }
338     {
339         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
340         { \@@_error:n { tabularnote~forbidden } }
341         {

```

`\l_@@_number_of_notes_int` is used to count the number of successive tabular notes such as in `\tabularnote{Note 1}\tabularnote{Note 2}\tabularnote{Note 3}`. We will have to compose the labels of these notes as a comma separated list (e.g. a,b,c).

```

342         \int_incr:N \l_@@_number_of_notes_int

```

We expand the content of the note at the point of use of `\tabularnote` as does `\footnote`.

```

343         \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
344         \peek_meaning:NF \tabularnote
345         {

```

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

```

346         \hbox_set:Nn \l_tmpa_box
347         {

```

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

```

348         \@@_notes_label_in_tabular:n
349         {
350             \stepcounter { tabularnote }
351             \@@_notes_style:n { tabularnote }
352             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
353             {
354                 ,
355                 \stepcounter { tabularnote }
356                 \@@_notes_style:n { tabularnote }
357             }
358         }
359     }

```

We use `\refstepcounter` in order to have the (last) tabular note referenceable (with the standard command `\label`) and that's why we have to go back with a decrementation of the counter `tabularnote` first.

```

360         \addtocounter { tabularnote } { -1 }
361         \refstepcounter { tabularnote }
362         \int_zero:N \l_@@_number_of_notes_int
363         \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).

```

364         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
365     }
366 }
367 }
368 }
369 }

```

³⁹We should try to find a solution to that problem.

Command for creation of rectangle nodes

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

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

```

370 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
371 {
372   \begin { pgfscope }
373   \pgfset
374   {
375     outer~sep = \c_zero_dim ,
376     inner~sep = \c_zero_dim ,
377     minimum~size = \c_zero_dim
378   }
379   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
380   \pgfnode
381   { rectangle }
382   { center }
383   {
384     \vbox_to_ht:nn
385     { \dim_abs:n { #5 - #3 } }
386     {
387       \vfill
388       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
389     }
390   }
391   { #1 }
392   { }
393   \end { pgfscope }
394 }
```

The command `\@@_pgf_rect_node:nnn` is a variant of `\@@_pgr_rect_node:nnnn`: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

```

395 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
396 {
397   \begin { pgfscope }
398   \pgfset
399   {
400     outer~sep = \c_zero_dim ,
401     inner~sep = \c_zero_dim ,
402     minimum~size = \c_zero_dim
403   }
404   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
405   \pgfpointdiff { #3 } { #2 }
406   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
407   \pgfnode
408   { rectangle }
409   { center }
410   {
411     \vbox_to_ht:nn
412     { \dim_abs:n \l_tmpb_dim }
413     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
414   }
415   { #1 }
416   { }
417   \end { pgfscope }
418 }
```

The options

By default, the commands `\cellcolor` and `\rowcolor` are available for the user in the cells of the tabular (the user may use the commands provided by `\colortbl`). However, if the key `colortbl-like` is used, these commands are available.

```
419 \bool_new:N \l_@@_colortbl_like_bool
```

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

```
420 \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`).

```
421 \dim_new:N \l_@@_cell_space_top_limit_dim
422 \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.

```
423 \dim_new:N \l_@@_inter_dots_dim
424 \dim_set:Nn \l_@@_inter_dots_dim { 0.45 em }
```

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).

```
425 \dim_new:N \l_@@_xdots_shorten_dim
426 \dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 em }
```

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.

```
427 \dim_new:N \l_@@_radius_dim
428 \dim_set:Nn \l_@@_radius_dim { 0.53 pt }
```

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.

```
429 \tl_new:N \l_@@_xdots_line_style_tl
430 \tl_const:Nn \c_@@_standard_tl { standard }
431 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
432 \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).

```
433 \tl_new:N \l_@@_baseline_tl
434 \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`).

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

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

```
436 \bool_new:N \l_@@_parallelize_diags_bool
437 \bool_set_true:N \l_@@_parallelize_diags_bool
```

If the flag `\l_@@_vlines_bool` is raised, horizontal space will be reserved in the the preamble of the array (for the vertical rules) and, after the construction of the array, the vertical rules will be drawn.

```
438 \bool_new:N \l_@@_vlines_bool
```

If the flag `\l_@@_hlines_bool` is raised, vertical space will be reserved between the rows of the array (for the horizontal rules) and, after the construction of the array, the vertical rules will be drawn.

```
439 \bool_new:N \l_@@_hlines_bool
```

The flag `\l_@@_except_corners_bool` will be raised when the key `except-corners` will be used. In that case, the corners will be computed before we draw rules and the rules won't be drawn in the corners. As expected, the key `hvlines-except-corners` raises the key `except-corners`.

```
440 \clist_new:N \l_@@_except_corners_clist
441 \dim_new:N \l_@@_notes_above_space_dim
442 \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm }
```

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.).

```
443 \bool_new:N \l_@@_nullify_dots_bool
```

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).

```
444 \bool_new:N \l_@@_auto_columns_width_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.

```
445 \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”.

```
446 \bool_new:N \l_@@_medium_nodes_bool
447 \bool_new:N \l_@@_large_nodes_bool
```

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

```
448 \dim_new:N \l_@@_left_margin_dim
449 \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`.

```
450 \dim_new:N \l_@@_extra_left_margin_dim
451 \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.

```
452 \tl_new:N \l_@@_end_of_row_tl
453 \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 “:”.

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

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

```
455 \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 `max-delimiter-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.

```
456 \bool_new:N \l_@@_max_delimiter_width_bool
```

```
457 \keys_define:nn { NiceMatrix / xdots }
458 {
459   line-style .code:n =
460   {
461     \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.

```
462     { \cs_if_exist_p:N \tikzpicture }
463     { \str_if_eq_p:nn { #1 } { standard } }
464     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
465     { \@@_error:n { bad-option-for~line-style } }
466   } ,
467   line-style .value_required:n = true ,
468   color .tl_set:N = \l_@@_xdots_color_tl ,
469   color .value_required:n = true ,
470   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
471   shorten .value_required:n = true ,
```

The options `down` and `up` are not documented for the final user because he should use the syntax with `^` and `_`.

```
472   down .tl_set:N = \l_@@_xdots_down_tl ,
473   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).

```
474   draw-first .code:n = \prg_do_nothing: ,
475   unknown .code:n = \@@_error:n { Unknown-option-for~xdots }
476 }
```

```
477 \keys_define:nn { NiceMatrix / rules }
478 {
479   color .tl_set:N = \l_@@_rules_color_tl ,
480   color .value_required:n = true ,
481   width .dim_set:N = \arrayrulewidth ,
482   width .value_required:n = true
483 }
```

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

```
484 \keys_define:nn { NiceMatrix / Global }
485 {
```

```

486 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
487 standard-cline .default:n = true ,
488 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
489 cell-space-top-limit .value_required:n = true ,
490 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
491 cell-space-bottom-limit .value_required:n = true ,
492 cell-space-limits .code:n =
493 {
494     \dim_set:Nn \l_@@_cell_space_bottom_limit_dim { #1 }
495     \dim_set:Nn \l_@@_cell_space_top_limit_dim { #1 }
496 } ,
497 cell-spaces-limits .value_required:n = true ,
498 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
499 max-delimiter-width .bool_set:N = \l_@@_max_delimiter_width_bool ,
500 light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
501 light-syntax .default:n = true ,
502 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
503 end-of-row .value_required:n = true ,
504 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
505 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
506 last-row .int_set:N = \l_@@_last_row_int ,
507 last-row .default:n = -1 ,
508 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
509 code-for-first-col .value_required:n = true ,
510 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
511 code-for-last-col .value_required:n = true ,
512 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
513 code-for-first-row .value_required:n = true ,
514 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
515 code-for-last-row .value_required:n = true ,
516 hlines .bool_set:N = \l_@@_hlines_bool ,
517 vlines .bool_set:N = \l_@@_vlines_bool ,
518 hvlines .code:n =
519 {
520     \bool_set_true:N \l_@@_vlines_bool
521     \bool_set_true:N \l_@@_hlines_bool
522 } ,
523 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.

```

524 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
525 renew-dots .value_forbidden:n = true ,
526 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
527 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
528 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
529 create-extra-nodes .meta:n =
530 { create-medium-nodes , create-large-nodes } ,
531 left-margin .dim_set:N = \l_@@_left_margin_dim ,
532 left-margin .default:n = \arraycolsep ,
533 right-margin .dim_set:N = \l_@@_right_margin_dim ,
534 right-margin .default:n = \arraycolsep ,
535 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
536 margin .default:n = \arraycolsep ,
537 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
538 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
539 extra-margin .meta:n =
540 { extra-left-margin = #1 , extra-right-margin = #1 } ,
541 extra-margin .value_required:n = true ,
542 }

```

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

```

543 \keys_define:nn { NiceMatrix / Env }
544 {
545   except-corners .clist_set:N = \l_@@_except_corners_clist ,
546   except-corners .default:n = { NW , SW , NE , SE } ,
547   hvlines-except-corners .code:n =
548   {
549     \clist_set:Nn \l_@@_except_corners_clist { #1 }
550     \bool_set_true:N \l_@@_vlines_bool
551     \bool_set_true:N \l_@@_hlines_bool
552   } ,
553   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
554   code-before .code:n =
555   {
556     \tl_if_empty:nF { #1 }
557     {
558       \tl_put_right:Nn \l_@@_code_before_tl { #1 }
559       \bool_set_true:N \l_@@_code_before_bool
560     }
561   } ,

```

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

```

562   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
563   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
564   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
565   baseline .tl_set:N = \l_@@_baseline_tl ,
566   baseline .value_required:n = true ,
567   columns-width .code:n =
568   { \tl_if_eq:nnTF { #1 } { auto }
569     { \bool_set_true:N \l_@@_auto_columns_width_bool }
570     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
571   columns-width .value_required:n = true ,
572   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.

```

573   \legacy_if:nF { measuring@ }
574   {
575     \str_set:Nn \l_tmpa_str { #1 }
576     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
577     { \@@_error:nn { Duplicate-name } { #1 } }
578     { \seq_gput_left:N \g_@@_names_seq \l_tmpa_str }
579     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
580   } ,
581   name .value_required:n = true ,
582   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
583   code-after .value_required:n = true ,
584   colortbl-like .code:n =
585   { \bool_set_true:N \l_@@_colortbl_like_bool
586     \bool_set_true:N \l_@@_code_before_bool ,
587   colortbl-like .value_forbidden:n = true
588   }
589 \keys_define:nn { NiceMatrix / notes }
590 {
591   para .bool_set:N = \l_@@_notes_para_bool ,
592   para .default:n = true ,
593   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
594   code-before .value_required:n = true ,
595   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
596   code-after .value_required:n = true ,
597   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
598   bottomrule .default:n = true ,
599   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,

```

```

600 style .value_required:n = true ,
601 label-in-tabular .code:n =
602   \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
603 label-in-tabular .value_required:n = true ,
604 label-in-list .code:n =
605   \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
606 label-in-list .value_required:n = true ,
607 enumitem-keys .code:n =
608   {
609     \bool_if:NTF \c_@@_in_preamble_bool
610     {
611       \AtBeginDocument
612       {
613         \bool_if:NT \c_@@_enumitem_loaded_bool
614         { \setlist* [ tabularnotes ] { #1 } }
615       }
616     }
617     {
618       \bool_if:NT \c_@@_enumitem_loaded_bool
619       { \setlist* [ tabularnotes ] { #1 } }
620     }
621   } ,
622 enumitem-keys .value_required:n = true ,
623 enumitem-keys-para .code:n =
624   {
625     \bool_if:NTF \c_@@_in_preamble_bool
626     {
627       \AtBeginDocument
628       {
629         \bool_if:NT \c_@@_enumitem_loaded_bool
630         { \setlist* [ tabularnotes* ] { #1 } }
631       }
632     }
633     {
634       \bool_if:NT \c_@@_enumitem_loaded_bool
635       { \setlist* [ tabularnotes* ] { #1 } }
636     }
637   } ,
638 enumitem-keys-para .value_required:n = true ,
639 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
640 }

```

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

```

641 \keys_define:nn { NiceMatrix }
642 {
643   NiceMatrixOptions .inherit:n =
644     { NiceMatrix / Global } ,
645   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
646   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
647   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
648   NiceMatrix .inherit:n =
649     {
650       NiceMatrix / Global ,
651       NiceMatrix / Env ,
652     } ,
653   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
654   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
655   NiceTabular .inherit:n =
656     {
657       NiceMatrix / Global ,
658       NiceMatrix / Env
659     } ,

```

```

660 NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
661 NiceTabular / rules .inherit:n = NiceMatrix / rules ,
662 NiceArray .inherit:n =
663 {
664     NiceMatrix / Global ,
665     NiceMatrix / Env ,
666 } ,
667 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
668 NiceArray / rules .inherit:n = NiceMatrix / rules ,
669 pNiceArray .inherit:n =
670 {
671     NiceMatrix / Global ,
672     NiceMatrix / Env ,
673 } ,
674 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
675 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
676 }

```

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

```

677 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
678 {
679     delimiters-color .tl_set:N = \l_@@_delimiters_color_tl ,
680     delimiters-color .value_required:n = true ,
681     last-col .code:n = \tl_if_empty:nF { #1 }
682         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
683         \int_zero:N \l_@@_last_col_int ,
684     small .bool_set:N = \l_@@_small_bool ,
685     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.

```

686     renew-matrix .code:n = \@@_renew_matrix: ,
687     renew-matrix .value_forbidden:n = true ,

```

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

```

688     transparent .meta:n = { renew-dots , renew-matrix } ,
689     transparent .value_forbidden:n = true ,

```

The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

```

690     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.

```

691     columns-width .code:n =
692         \tl_if_eq:nnTF { #1 } { auto }
693         { \@@_error:n { Option-auto-for-columns-width } }
694         { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

695     allow-duplicate-names .code:n =
696         \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
697     allow-duplicate-names .value_forbidden:n = true ,

```

By default, the specifier used in the preamble of the array (for example in `{pNiceArray}`) to draw a vertical dotted line between two columns is the colon “:”. However, it’s possible to change this letter

with `letter-for-dotted-lines` and, by the way, the letter “:” will remain free for other packages (for example `arydshln`).

```

698   letter-for-dotted-lines .code:n =
699   {
700     \tl_if_single_token:nTF { #1 }
701     { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
702     { \@@_error:n { Bad-value-for-letter-for-dotted-lines } }
703   } ,
704   letter-for-dotted-lines .value_required:n = true ,
705   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
706   notes .value_required:n = true ,
707   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
708 }
709 \str_new:N \l_@@_letter_for_dotted_lines_str
710 \str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str

```

`\NiceMatrixOptions` is the command of the `nicematrix` package to fix options at the document level. The scope of these specifications is the current TeX group.

```

711 \NewDocumentCommand \NiceMatrixOptions { m }
712 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

713 \keys_define:nn { NiceMatrix / NiceMatrix }
714 {
715   last-col .code:n = \tl_if_empty:nTF {#1}
716   {
717     \bool_set_true:N \l_@@_last_col_without_value_bool
718     \int_set:Nn \l_@@_last_col_int { -1 }
719   }
720   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
721   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
722   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
723   small .bool_set:N = \l_@@_small_bool ,
724   small .value_forbidden:n = true ,
725   delimiters-color .tl_set:N = \l_@@_delimiters_color_tl ,
726   delimiters-color .value_required:n = true ,
727   unknown .code:n = \@@_error:n { Unknown-option-for-NiceMatrix }
728 }

```

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

```

729 \keys_define:nn { NiceMatrix / NiceArray }
730 {

```

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.

```

731   small .bool_set:N = \l_@@_small_bool ,
732   small .value_forbidden:n = true ,
733   last-col .code:n = \tl_if_empty:nF { #1 }
734   { \@@_error:n { last-col-non-empty-for-NiceArray } }
735   \int_zero:N \l_@@_last_col_int ,
736   notes / para .bool_set:N = \l_@@_notes_para_bool ,
737   notes / para .default:n = true ,
738   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
739   notes / bottomrule .default:n = true ,
740   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
741   tabularnote .value_required:n = true ,
742   delimiters-color .tl_set:N = \l_@@_delimiters_color_tl ,
743   delimiters-color .value_required:n = true ,
744   unknown .code:n = \@@_error:n { Unknown-option-for-NiceArray }
745 }

```

```

746 \keys_define:nn { NiceMatrix / pNiceArray }
747 {
748   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
749   last-col .code:n = \tl_if_empty:nF {#1}
750     { \@@_error:n { last-col~non-empty~for~NiceArray } }
751     \int_zero:N \l_@@_last_col_int ,
752   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
753   small .bool_set:N = \l_@@_small_bool ,
754   small .value_forbidden:n = true ,
755   unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
756 }

```

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

```

757 \keys_define:nn { NiceMatrix / NiceTabular }
758 {
759   notes / para .bool_set:N = \l_@@_notes_para_bool ,
760   notes / para .default:n = true ,
761   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
762   notes / bottomrule .default:n = true ,
763   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
764   tabularnote .value_required:n = true ,
765   last-col .code:n = \tl_if_empty:nF {#1}
766     { \@@_error:n { last-col~non-empty~for~NiceArray } }
767     \int_zero:N \l_@@_last_col_int ,
768   unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular }
769 }

```

Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_Cell:–\@@_end_Cell:` will be used to format the cells of the array. In the code, the affectations are global because this pseudo-environment will be used in the cells of a `\halign` (via an environment {array}).

```

770 \cs_new_protected:Npn \@@_Cell:
771 {

```

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

```

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

```

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

```

773   \int_gincr:N \c@jCol

```

Now, we increment the counter of the rows. We don’t do this incrementation in the `\everycr` because some packages, like `arydshln`, create special rows in the `\halign` that we don’t want to take into account.

```

774   \int_compare:nNnT \c@jCol = 1
775     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
776   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }

```

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

```

777   \hbox_set:Nw \l_@@_cell_box
778   \bool_if:NF \l_@@_NiceTabular_bool
779   {
780     \c_math_toggle_token
781     \bool_if:NT \l_@@_small_bool \scriptstyle
782   }

```

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.

```

783 \int_compare:nNnTF \c@iRow = 0
784 {
785   \int_compare:nNnT \c@jCol > 0
786   {
787     \l_@@_code_for_first_row_tl
788     \xglobal \colorlet { nicematrix-first-row } { . }
789   }
790 }
791 {
792   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
793   {
794     \l_@@_code_for_last_row_tl
795     \xglobal \colorlet { nicematrix-last-row } { . }
796   }
797 }
798 }
```

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.

```

799 \cs_new_protected:Npn \@@_begin_of_row:
800 {
801   \int_gincr:N \c@iRow
802   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
803   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
804   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
805   \pgfpicture
806   \pgfrememberpicturepositiononpagetrue
807   \pgfcoordinate
808   { \@@_env: - row - \int_use:N \c@iRow - base }
809   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
810   \str_if_empty:NF \l_@@_name_str
811   {
812     \pgfnodealias
813     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
814     { \@@_env: - row - \int_use:N \c@iRow - base }
815   }
816   \endpgfpicture
817 }
```

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.

```

818 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
819 {
820   \int_compare:nNnTF \c@iRow = 0
821   {
822     \dim_gset:Nn \g_@@_dp_row_zero_dim
823     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
824     \dim_gset:Nn \g_@@_ht_row_zero_dim
825     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
826   }
827   {
828     \int_compare:nNnT \c@iRow = 1
829     {
830       \dim_gset:Nn \g_@@_ht_row_one_dim
831       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
832     }
833   }
```

```

833     }
834 }

835 \cs_new_protected:Npn \@@_rotate_cell_box:
836 {
837     \box_rotate:Nn \l_@@_cell_box { 90 }
838     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
839     {
840         \vbox_set_top:Nn \l_@@_cell_box
841         {
842             \vbox_to_zero:n { }
843             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
844             \box_use:N \l_@@_cell_box
845         }
846     }
847     \bool_gset_false:N \g_@@_rotate_bool
848 }

849 \cs_new_protected:Npn \@@_adjust_size_box:
850 {
851     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
852     {
853         \box_set_wd:Nn \l_@@_cell_box
854         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
855         \dim_gzero:N \g_@@_blocks_wd_dim
856     }
857     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
858     {
859         \box_set_dp:Nn \l_@@_cell_box
860         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
861         \dim_gzero:N \g_@@_blocks_dp_dim
862     }
863     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
864     {
865         \box_set_ht:Nn \l_@@_cell_box
866         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
867         \dim_gzero:N \g_@@_blocks_ht_dim
868     }
869 }

870 \cs_new_protected:Npn \@@_end_Cell:
871 {
872     \@@_math_toggle_token:
873     \hbox_set_end:
874     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
875     \@@_adjust_size_box:

876     \box_set_ht:Nn \l_@@_cell_box
877     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
878     \box_set_dp:Nn \l_@@_cell_box
879     { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }

```

We want to compute in $\text{\g_@@_max_cell_width_dim}$ the width of the widest cell of the array (except the cells of the “first column” and the “last column”).

```

880     \dim_gset:Nn \g_@@_max_cell_width_dim
881     { \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”.

```

882     \@@_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. As of now, we use the following technic:

- if the width of the box `\l_@@_cell_box` (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a `\rlap`, a `\llap` or a `\mathclap` of `mathtools`).
- the cells with a command `\Ldots` or `\Cdots`, `\Vdots`, etc., should also be considered as empty; if `nullify-dots` is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of `code-after`); 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.

```

883 \bool_if:NTF \g_@@_empty_cell_bool
884 { \box_use_drop:N \l_@@_cell_box }
885 {
886   \bool_lazy_or:nnTF
887     \g_@@_not_empty_cell_bool
888     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
889     \@@_node_for_the_cell:
890     { \box_use_drop:N \l_@@_cell_box }
891 }
892 \bool_gset_false:N \g_@@_empty_cell_bool
893 \bool_gset_false:N \g_@@_not_empty_cell_bool
894 }
```

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

```

895 \cs_new_protected:Npn \@@_node_for_the_cell:
896 {
897   \pgfpicture
898   \pgfsetbaseline \c_zero_dim
899   \pgfrememberpicturepositiononpagetrue
900   \pgfset
901   {
902     inner~sep = \c_zero_dim ,
903     minimum~width = \c_zero_dim
904   }
905   \pgfnode
906   { rectangle }
907   { base }
908   { \box_use_drop:N \l_@@_cell_box }
909   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
910   { }
911   \str_if_empty:NF \l_@@_name_str
912   {
913     \pgfnodealias
914     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
915     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
916   }
917   \endpgfpicture
918 }
```

The second argument of the following command `\@@_instruction_of_type:nnn` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). The third argument is the list of options. This command writes in the corresponding `\g_@@_type_lines_tl` the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots[color=red] & & 
\end{pNiceMatrix}
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & \cdots & & 6 \\ 7 & \cdots & & \end{pmatrix}$$

the content of `\g_@@_Cdots_lines_tl` will be:

```
\@@_draw_Cdots:nnn {2}{2}{}  
\@@_draw_Cdots:nnn {3}{2}{color=red}
```

The first argument is a boolean which indicates whether you must put the instruction on the left or on the right on the list of instructions.

```
919 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3  
920 {
```

It's important to use a `\tl_gput_right:cx` and not a `\tl_gput_left:cx` because we want the `\Ddots` lines to be drawn in the order of appearance in the array (for parallelisation).

```
921   \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx  
922   { g_@@_ #2 _ lines _ tl }  
923   {  
924     \use:c { @@ _ draw _ #2 : nnn }  
925     { \int_use:N \c@iRow }  
926     { \int_use:N \c@jCol }  
927     { \exp_not:n { #3 } }  
928   }  
929 }
```

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

```
930 \cs_new_protected:Npn \@@_revtex_array:  
931 {  
932   \cs_set_eq:NN \@acoll \@arrayacol  
933   \cs_set_eq:NN \@acolr \@arrayacol  
934   \cs_set_eq:NN \@acol \@arrayacol  
935   \cs_set_nopar:Npn \@halignto { }  
936   \@array@array  
937 }  
  
938 \cs_new_protected:Npn \@@_array:  
939 {  
940   \bool_if:NTF \c_@@_revtex_bool  
941   \@@_revtex_array:  
942   {  
943     \bool_if:NTF \l_@@_NiceTabular_bool  
944     { \dim_set_eq:NN \col@sep \tabcolsep }  
945     { \dim_set_eq:NN \col@sep \arraycolsep }  
946     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim  
947     { \cs_set_nopar:Npn \@halignto { } }  
948     { \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`.

```
949   \@tabarray  
950 }
```

`\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.

```
951   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]  
952 }
```

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.

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

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

```
954 \cs_new_protected:Npn \@@_create_row_node:  
955 {
```

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

```

956   \hbox
957   {
958     \bool_if:NT \l_@@_code_before_bool
959     {
960       \vtop
961       {
962         \skip_vertical:N 0.5\arrayrulewidth
963         \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
964         \skip_vertical:N -0.5\arrayrulewidth
965       }
966     }
967     \pgfpicture
968     \pgfrememberpicturerepositiononpagetrue
969     \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
970     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
971     \str_if_empty:NF \l_@@_name_str
972     {
973       \pgfnodealias
974       { \l_@@_name_str - row - \int_use:N \c@iRow }
975       { \@@_env: - row - \int_use:N \c@iRow }
976     }
977     \endpgfpicture
978   }
979 }

```

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

```

980 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
981 \cs_new_protected:Npn \@@_everycr_i:
982 {
983   \int_gzero:N \c@jCol
984   \bool_gset_false:N \g_@@_after_col_zero_bool
985   \bool_if:NF \g_@@_row_of_col_done_bool
986   {
987     \@@_create_row_node:

```

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

```

988   \bool_if:NT \l_@@_hlines_bool
989   {

```

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.

```

990     \int_compare:nNnT \c@iRow > { -1 }
991     {
992       \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@`.

```

993       { \hrule height \arrayrulewidth width \c_zero_dim }
994     }
995   }
996 }
997 }

```

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`).

```

998 \cs_set_protected:Npn \@@_newcolumnntype #1
999 {
1000   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1001   \peek_meaning:NTF [

```

```

1002     { \newcol@ #1 }
1003     { \newcol@ #1 [ 0 ] }
1004 }

```

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

```

1005 \cs_set_protected:Npn \@@_renew_dots:
1006 {
1007     \cs_set_eq:NN \ldots \@@_Ldots
1008     \cs_set_eq:NN \cdots \@@_Cdots
1009     \cs_set_eq:NN \vdots \@@_Vdots
1010     \cs_set_eq:NN \ddots \@@_Ddots
1011     \cs_set_eq:NN \iddots \@@_Iddots
1012     \cs_set_eq:NN \dots \@@_Ldots
1013     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1014 }

```

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

```

1015 \cs_new_protected:Npn \@@_colortbl_like:
1016 {
1017     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1018     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1019     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1020 }

```

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

```

1021 \cs_new_protected:Npn \@@_pre_array:
1022 {

```

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 ⁴⁰.

```

1023     \bool_if:NT \c_@@_booktabs_loaded_bool
1024     { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1025     \box_clear_new:N \l_@@_cell_box
1026     \cs_if_exist:NT \theiRow
1027     { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1028     \int_gzero_new:N \c@iRow
1029     \cs_if_exist:NT \thejCol
1030     { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1031     \int_gzero_new:N \c@jCol
1032     \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}`).

```

1033     \bool_if:NT \l_@@_small_bool
1034     {
1035         \cs_set_nopar:Npn \arraystretch { 0.47 }
1036         \dim_set:Nn \arraycolsep { 1.45 pt }
1037     }

```

⁴⁰cf. `\nicematrix@redefine@check@rerun`

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`.

```

1038   \cs_set_nopar:Npn \ialign
1039   {
1040     \bool_if:NTF \c_@@_colortbl_loaded_bool
1041     {
1042       \CT@everycr
1043       {
1044         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1045         \@@_everycr:
1046       }
1047     }
1048     { \everycr { \@@_everycr: } }
1049     \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`⁴¹ 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`.

```

1050     \dim_gzero_new:N \g_@@_dp_row_zero_dim
1051     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1052     \dim_gzero_new:N \g_@@_ht_row_zero_dim
1053     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1054     \dim_gzero_new:N \g_@@_ht_row_one_dim
1055     \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1056     \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1057     \dim_gzero_new:N \g_@@_ht_last_row_dim
1058     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1059     \dim_gzero_new:N \g_@@_dp_last_row_dim
1060     \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`.

```

1061     \cs_set_eq:NN \ialign \@@_old_ialign:
1062     \halign
1063     }

```

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).

```

1064     \cs_set_eq:NN \@@_old_ldots \ldots
1065     \cs_set_eq:NN \@@_old_cdots \cdots
1066     \cs_set_eq:NN \@@_old_vdots \vdots
1067     \cs_set_eq:NN \@@_old_ddots \ddots
1068     \cs_set_eq:NN \@@_old_iddots \iddots
1069     \bool_if:NTF \l_@@_standard_cline_bool
1070     { \cs_set_eq:NN \cline \@@_standard_cline }
1071     { \cs_set_eq:NN \cline \@@_cline }
1072     \cs_set_eq:NN \Ldots \@@_Ldots
1073     \cs_set_eq:NN \Cdots \@@_Cdots
1074     \cs_set_eq:NN \Vdots \@@_Vdots
1075     \cs_set_eq:NN \Ddots \@@_Ddots
1076     \cs_set_eq:NN \Iddots \@@_Iddots
1077     \cs_set_eq:NN \hdottedline \@@_hdottedline:
1078     \cs_set_eq:NN \Hline \@@_Hline:
1079     \cs_set_eq:NN \Hspace \@@_Hspace:
1080     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1081     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:

```

⁴¹The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1082 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1083 \cs_set_eq:NN \Block \@@_Block:
1084 \cs_set_eq:NN \rotate \@@_rotate:
1085 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1086 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1087 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1088 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1089 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1090 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1091 \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:

```

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

```

1092 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1093 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1094 \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`).

```

1095 \int_gzero_new:N \g_@@_row_total_int

```

The counter `\c@jCol` will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter `\g_@@_col_total_int`. These counters are updated in the command `\@@_Cell:` executed at the beginning of each cell.

```

1096 \int_gzero_new:N \g_@@_col_total_int
1097 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1098 \@@_renew_NC@rewrite@S:
1099 \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.

```

1100 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1101 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1102 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1103 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1104 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1105 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1106 \tl_gclear_new:N \g_nicematrix_code_before_tl
1107 }

```

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

The environment `{NiceArrayWithDelims}`

```

1108 \NewDocumentEnvironment { NiceArrayWithDelims } { m m O { } m ! O { } }
1109 {
1110 \@@_provide_pgfsyspdfmark:
1111 \bool_if:NT \c_@@_footnote_bool \savenotes

```

The aim of the following `\bgroup` (the corresponding `\egroup` is, of course, at the end of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

```

1112 \bgroup
1113 \tl_set:Nn \l_@@_left_delim_tl { #1 }
1114 \tl_set:Nn \l_@@_right_delim_tl { #2 }
1115 \int_gzero:N \g_@@_block_box_int

```

```

1116 \dim_zero:N \g_@@_width_last_col_dim
1117 \dim_zero:N \g_@@_width_first_col_dim
1118 \bool_gset_false:N \g_@@_row_of_col_done_bool
1119 \str_if_empty:NT \g_@@_name_env_str
1120 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1121 \@@_adapt_S_column:
1122 \bool_if:NTF \l_@@_NiceTabular_bool
1123 \mode_leave_vertical:
1124 \@@_test_if_math_mode:
1125 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1126 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁴². 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.

```

1127 \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.

```

1128 \cs_if_exist:NT \tikz@library@external@loaded
1129 {
1130   \tikzexternaldisable
1131   \cs_if_exist:NT \ifstandalone
1132     { \tikzset { external / optimize = false } }
1133 }

```

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

```

1134 \int_gincr:N \g_@@_env_int
1135 \bool_if:NF \l_@@_block_auto_columns_width_bool
1136 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1137 \seq_gclear:N \g_@@_blocks_seq
1138 \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`.

```

1139 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1140 \seq_gclear:N \g_@@_pos_of_xdots_seq
1141 \tl_if_exist:cT { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
1142 {
1143   \bool_set_true:N \l_@@_code_before_bool
1144   \exp_args:NNv \tl_put_right:Nn \l_@@_code_before_tl
1145     { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
1146 }

```

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`.

```

1147 \bool_if:NTF \l_@@_NiceArray_bool
1148 { \keys_set:nn { NiceMatrix / NiceArray } }
1149 { \keys_set:nn { NiceMatrix / pNiceArray } }
1150 { #3 , #5 }

1151 \tl_if_empty:NF \l_@@_rules_color_tl
1152 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }

```

⁴²e.g. `\color[rgb]{0.5,0.5,0}`

If the key `code-before` is used, we have to create the `col` nodes and the `row` nodes before the creation of the array. First, we have to test whether the size of the array has been written in the `aux` file in a previous run. In this case, a command `\@@_size_nb_of_env:` has been created.

```

1153 \bool_if:NT \l_@@_code_before_bool
1154 {
1155     \seq_if_exist:cT { @@_size_ \int_use:N \g_@@_env_int _ seq }
1156     {

```

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

```

1157     \int_zero_new:N \c@iRow
1158     \int_set:Nn \c@iRow
1159         { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
1160     \int_zero_new:N \c@jCol
1161     \int_set:Nn \c@jCol
1162         { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 }

```

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

```

1163     \int_compare:nNnF \l_@@_last_row_int = { -2 }
1164     { \int_decr:N \c@iRow }
1165     \int_compare:nNnF \l_@@_last_col_int = { -2 }
1166     { \int_decr:N \c@jCol }

```

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

```

1167     \pgfsys@markposition { \@@_env: - position }
1168     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1169     \pgfpicture

```

First, the creation of the `row` nodes.

```

1170     \int_step_inline:nnn
1171         { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 1 }
1172         { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 + 1 }
1173         {
1174             \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1175             \pgfcoordinate { \@@_env: - row - ##1 }
1176                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1177         }

```

Now, the creation of the `col` nodes.

```

1178     \int_step_inline:nnn
1179         { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
1180         { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 + 1 }
1181         {
1182             \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1183             \pgfcoordinate { \@@_env: - col - ##1 }
1184                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1185         }
1186     \endpgfpicture
1187     \group_begin:
1188     \bool_if:NT \c_@@_tikz_loaded_bool
1189     {
1190         \tikzset
1191         {
1192             every-picture / .style =
1193                 { overlay , name-prefix = \@@_env: - }
1194         }
1195     }
1196     \cs_set_eq:NN \cellcolor \@@_cellcolor
1197     \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1198     \cs_set_eq:NN \rowcolor \@@_rowcolor
1199     \cs_set_eq:NN \rowcolors \@@_rowcolors
1200     \cs_set_eq:NN \columncolor \@@_columncolor

```

```
1201 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
```

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

```
1202 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1203 \seq_clear_new:N \l_@@_colors_seq
1204 \l_@@_code_before_tl
1205 \@@_actually_color:
1206 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1207 \group_end:
1208 }
1209 }
```

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

```
1210 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1211 {
1212   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1213   {
1214     \dim_gset:Nn \g_@@_ht_last_row_dim
1215     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1216     \dim_gset:Nn \g_@@_dp_last_row_dim
1217     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1218   }
1219 }
1220 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1221 {
1222   \bool_set_true:N \l_@@_last_row_without_value_bool
```

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

```
1223 \str_if_empty:NTF \l_@@_name_str
1224 {
1225   \cs_if_exist:cT { @@_last_row_ \int_use:N \g_@@_env_int }
1226   {
1227     \int_set:Nn \l_@@_last_row_int
1228     { \use:c { @@_last_row_ \int_use:N \g_@@_env_int } }
1229   }
1230 }
1231 {
1232   \cs_if_exist:cT { @@_last_row_ \l_@@_name_str }
1233   {
1234     \int_set:Nn \l_@@_last_row_int
1235     { \use:c { @@_last_row_ \l_@@_name_str } }
1236   }
1237 }
1238 }
```

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

```
1239 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1240 {
1241   \str_if_empty:NTF \l_@@_name_str
1242   {
1243     \cs_if_exist:cT { @@_last_col_ \int_use:N \g_@@_env_int }
1244     {
1245       \int_set:Nn \l_@@_last_col_int
1246       { \use:c { @@_last_col_ \int_use:N \g_@@_env_int } }
1247     }
1248   }
1249   {
1250     \cs_if_exist:cT { @@_last_col_ \l_@@_name_str }
1251     {
1252       \int_set:Nn \l_@@_last_col_int
1253       { \use:c { @@_last_col_ \l_@@_name_str } }
```

```

1254     }
1255   }
1256 }

```

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

```

1257 \@@_pre_array:

```

We compute the width of the two delimiters.

```

1258 \dim_zero_new:N \l_@@_left_delim_dim
1259 \dim_zero_new:N \l_@@_right_delim_dim
1260 \bool_if:NTF \l_@@_NiceArray_bool
1261 {
1262   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1263   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1264 }
1265 {

```

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

```

1266 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 #1 $ }
1267 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1268 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 #2 $ }
1269 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1270 }

```

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

```

1271 \box_clear_new:N \l_@@_the_array_box

```

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

```

1272 \bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:

```

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

```

1273 \@@_construct_preamble:n { #4 }

```

Now, the preamble is constructed in `\g_@@_preamble_tl`

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).

```

1274 \hbox_set:Nw \l_@@_the_array_box
1275 \skip_horizontal:N \l_@@_left_margin_dim
1276 \skip_horizontal:N \l_@@_extra_left_margin_dim
1277 \c_math_toggle_token
1278 \bool_if:NTF \l_@@_light_syntax_bool
1279 { \use:c { @@-light-syntax } }
1280 { \use:c { @@-normal-syntax } }
1281 }
1282 {
1283   \bool_if:NTF \l_@@_light_syntax_bool
1284   { \use:c { end @@-light-syntax } }
1285   { \use:c { end @@-normal-syntax } }
1286   \c_math_toggle_token
1287   \skip_horizontal:N \l_@@_right_margin_dim
1288   \skip_horizontal:N \l_@@_extra_right_margin_dim
1289   \hbox_set_end:

```

End of the construction of the array (in the box `\l_@@_the_array_box`).

If the user has used the key `last-row` with a value, we control that the given value is correct (since we have just constructed the array, we know the real number of rows of the array).

```

1290 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1291 {
1292   \bool_if:NF \l_@@_last_row_without_value_bool
1293   {
1294     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1295     {
1296       \@@_error:n { Wrong~last~row }
1297       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1298     }
1299   }
1300 }

```

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”.⁴³

```

1301 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1302 \bool_if:nTF \g_@@_last_col_found_bool
1303 { \int_gdecr:N \c@jCol }
1304 {
1305   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1306   { \@@_error:n { last~col~not~used } }
1307 }

```

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

```

1308 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1309 \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. 94).

```

1310 \int_compare:nNnT \l_@@_first_col_int = 0
1311 {
1312   \skip_horizontal:N \col@sep
1313   \skip_horizontal:N \g_@@_width_first_col_dim
1314 }

```

The construction of the real box is different when `\l_@@_NiceArray_bool` is true (`{NiceArray}` or `{NiceTabular}`) and in the other environments because, in `{NiceArray}` or `{NiceTabular}`, we have no delimiter to put (but we have tabular notes to put). We begin with this case.

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

```

1315 \bool_if:NTF \l_@@_NiceArray_bool
1316 {
1317   \str_case:VnF \l_@@_baseline_tl
1318   {
1319     b \@@_use_arraybox_with_notes_b:
1320     c \@@_use_arraybox_with_notes_c:
1321   }
1322   \@@_use_arraybox_with_notes:
1323 }

```

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).

```

1324 {
1325   \int_compare:nNnTF \l_@@_first_row_int = 0
1326   {
1327     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1328     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1329   }
1330   { \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”.⁴⁴

⁴³We remind that the potential “first column” (exterior) has the number 0.

⁴⁴A 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).

```

1331 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1332 {
1333   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1334   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1335 }
1336 { \dim_zero:N \l_tmpb_dim }
1337 \hbox_set:Nn \l_tmpa_box
1338 {
1339   \c_math_toggle_token
1340   \tl_if_empty:NF \l_@@_delimiters_color_tl
1341   { \color { \l_@@_delimiters_color_tl } }
1342   \left #1
1343   \vcenter
1344   {

```

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.

```

1345     \skip_vertical:N -\l_tmpa_dim
1346     \skip_vertical:N -\arrayrulewidth
1347     \hbox
1348     {
1349       \bool_if:NTF \l_@@_NiceTabular_bool
1350       { \skip_horizontal:N -\tabcolsep }
1351       { \skip_horizontal:N -\arraycolsep }
1352       \@@_use_arraybox_with_notes_c:
1353       \bool_if:NTF \l_@@_NiceTabular_bool
1354       { \skip_horizontal:N -\tabcolsep }
1355       { \skip_horizontal:N -\arraycolsep }
1356     }

```

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

```

1357     \skip_vertical:N -\l_tmpb_dim
1358     \skip_vertical:N \arrayrulewidth
1359   }
1360   \right #2
1361   \c_math_toggle_token
1362 }

```

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 `max-delimiter-width` is used.

```

1363 \bool_if:NTF \l_@@_max_delimiter_width_bool
1364 { \@@_put_box_in_flow_bis:nn { #1 } { #2 } }
1365 \@@_put_box_in_flow:
1366 }

```

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. 95).

```

1367 \bool_if:NT \g_@@_last_col_found_bool
1368 {
1369   \skip_horizontal:N \g_@@_width_last_col_dim
1370   \skip_horizontal:N \col@sep
1371 }
1372 \bool_if:NF \l_@@_Matrix_bool
1373 {
1374   \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1375   { \@@_error:n { columns-not-used } }
1376 }
1377 \group_begin:
1378 \globaldefs = 1
1379 \@@_msg_redirect_name:nn { columns-not-used } { error }
1380 \group_end:
1381 \@@_after_array:

```

The aim of the following `\egroup` (the corresponding `\bgroup` is, of course, at the beginning of the environment) is to be able to put an exposant to a matrix in a mathematical formula.


```

1382   \egroup
1383   \bool_if:NT \c_@@_footnote_bool \endsavenotes
1384 }

```

This is the end of the environment `{NiceArrayWithDelims}`.

We construct the preamble of the array

The transformation of the preamble is an operation in several steps.

The argument of `\@@_construct_preamble:n` is the preamble as given by the final user to the environment `{NiceTabular}` (or a variant). The preamble will be constructed in `\g_@@_preamble_tl`.

```

1385 \cs_new_protected:Npn \@@_construct_preamble:n #1
1386 {

```

First, we will do an “expansion” of the preamble with the tools of the package `array` itself. This “expansion” will expand all the constructions with `*` and with all column types (defined by the user or by various packages using `\newcolumntype`).

Since we use the tools of `array` to do this expansion, we will have a programming which is not in the style of `expl3`.

We redefine the column types `w` and `W`. We use `\@@_newcolumntype` instead of `\newcolumntype` because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters `w` and `W`. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```

1387   \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).

```

1388   \bool_if:NTF \l_@@_Matrix_bool
1389   { \tl_gset:Nn \g_@@_preamble_tl { #1 } }
1390   {
1391     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1392     \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

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

```

1393     \@temptokena { #1 }

```

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

```

1394     \@tempswatruue

```

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

```

1395     \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

Now, we have to “patch” that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the “expansion”) following by a marker `\q_stop` and we will consume these tokens constructing the (new form of the) preamble in `\g_@@_preamble_tl`. This is done recursively with the command `\@@_patch_preamble:n`. In the same time, we will count the columns with the counter `\c@jCol`.

```

1396     \int_gzero_new:N \c@jCol
1397     \bool_if:NTF \l_@@_vlines_bool
1398     {
1399       \tl_gset:Nn \g_@@_preamble_tl
1400       { ! { \skip_horizontal:N \arrayrulewidth } }
1401     }
1402     { \tl_gclear:N \g_@@_preamble_tl }

```

The counter `\l_tmpa_int` will count the number of consecutive occurrences of the symbole `l`.

```

1403     \int_zero:N \l_tmpa_int

```

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

```

1404     \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1405     \int_gset_eq:NN \g_@@_static_num_of_col_int \c_jCol
1406 }

```

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

```

1407 \bool_if:NT \l_@@_colortbl_like_bool
1408 {
1409     \regex_replace_all:NnN
1410     \c_@@_columncolor_regex
1411     { \c { \@@_columncolor_preamble } }
1412     \g_@@_preamble_tl
1413 }

```

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

```

1414 \int_compare:nNnTF \l_@@_first_col_int = 0
1415 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1416 {
1417     \bool_lazy_all:nT
1418     {
1419         \l_@@_NiceArray_bool
1420         { \bool_not_p:n \l_@@_NiceTabular_bool }
1421         { \bool_not_p:n \l_@@_vlines_bool }
1422         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1423     }
1424     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1425 }
1426 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1427 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1428 {
1429     \bool_lazy_all:nT
1430     {
1431         \l_@@_NiceArray_bool
1432         { \bool_not_p:n \l_@@_NiceTabular_bool }
1433         { \bool_not_p:n \l_@@_vlines_bool }
1434         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1435     }
1436     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1437 }

```

We add a last column to raise a good error message when the user put more columns than allowed by its preamble. However, for technical reasons, it’s not possible to do that in `{\NiceTabular*}` (`\l_@@_tabular_width_dim=0pt`).

```

1438 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1439 {
1440     \tl_gput_right:Nn
1441     \g_@@_preamble_tl
1442     { > { \@@_error_too_much_cols: } 1 }
1443 }

```

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

```

1444 \group_end:
1445 }

```

```

1446 \cs_new_protected:Npn \@@_patch_preamble:n #1
1447 {
1448     \str_case:nnF { #1 }
1449     {
1450         c { \@@_patch_preamble_i:n #1 }
1451         l { \@@_patch_preamble_i:n #1 }
1452         r { \@@_patch_preamble_i:n #1 }

```

```

1453 > { \@@_patch_preamble_ii:nn #1 }
1454 ! { \@@_patch_preamble_ii:nn #1 }
1455 @ { \@@_patch_preamble_ii:nn #1 }
1456 | { \@@_patch_preamble_iii:n #1 }
1457 p { \@@_patch_preamble_iv:nnn t #1 }
1458 m { \@@_patch_preamble_iv:nnn c #1 }
1459 b { \@@_patch_preamble_iv:nnn b #1 }
1460 \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
1461 \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1462 \@@_true_c: { \@@_patch_preamble_vi:n #1 }
1463 C { \@@_error:nn { old~column~type } C }
1464 L { \@@_error:nn { old~column~type } L }
1465 R { \@@_error:nn { old~column~type } R }
1466 \q_stop { }
1467 }
1468 {
1469 \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1470 { \@@_patch_preamble_vii:n #1 }
1471 { \@@_fatal:nn { unknown~column~type } { #1 } }
1472 }
1473 }

```

For c, l and r

```

1474 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1475 {
1476 \tl_gput_right:Nn \g_@@_preamble_tl
1477 {
1478 > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } }
1479 #1
1480 < \@@_end_Cell:
1481 }

```

We increment the counter of columns.

```

1482 \int_gincr:N \c@jCol
1483 \@@_patch_preamble_viii:n
1484 }

```

For >, ! and @

```

1485 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1486 {
1487 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1488 \@@_patch_preamble:n
1489 }

```

For |

```

1490 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1491 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1492 \int_incr:N \l_tmpa_int
1493 \@@_patch_preamble_iii_i:n
1494 }
1495 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1496 {
1497 \str_if_eq:nnTF { #1 } |
1498 { \@@_patch_preamble_iii:n | }
1499 {
1500 \tl_gput_right:Nx \g_@@_preamble_tl
1501 {
1502 \exp_not:N !
1503 {
1504 \skip_horizontal:n
1505 {

```

```

1506         \dim_eval:n
1507         {
1508             \arrayrulewidth * \l_tmpa_int
1509             + \doublerulesep * ( \l_tmpa_int - 1)
1510         }
1511     }
1512 }
1513 }
1514 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1515 { \@@_vline:nn { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } }
1516 \int_zero:N \l_tmpa_int
1517 \@@_patch_preamble:n #1
1518 }
1519 }

```

For p, m and b

```

1520 \cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
1521 {
1522     \tl_gput_right:Nn \g_@@_preamble_tl
1523     {
1524         > {
1525             \@@_Cell:
1526             \begin { minipage } [ #1 ] { #3 }
1527             \mode_leave_vertical:
1528             \arraybackslash % added in the version 5.8
1529             \box_use:N \@@arstrutbox
1530         }
1531         c
1532         < { \box_use:N \@@arstrutbox \end { minipage } \@@_end_Cell: }
1533     }

```

We increment the counter of columns.

```

1534     \int_gincr:N \c@jCol
1535     \@@_patch_preamble_viii:n
1536 }

```

For w and W

```

1537 \cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
1538 {
1539     \tl_gput_right:Nn \g_@@_preamble_tl
1540     {
1541         > {
1542             \hbox_set:Nw \l_@@_cell_box
1543             \@@_Cell:
1544             \tl_set:Nn \l_@@_cell_type_tl { #3 }
1545         }
1546         c
1547         < {
1548             \@@_end_Cell:
1549             #1
1550             \hbox_set_end:
1551             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1552             \@@_adjust_size_box:
1553             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
1554         }
1555     }

```

We increment the counter of columns.

```

1556     \int_gincr:N \c@jCol
1557     \@@_patch_preamble_viii:n
1558 }

```

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

```

1559 \cs_new_protected:Npn \@@_patch_preamble_vi:n #1

```

```

1560 {
1561   \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

We increment the counter of columns.

```

1562   \int_gincr:N \c@jCol
1563   \@@_patch_preamble_viii:n
1564 }
1565 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
1566 {
1567   \tl_gput_right:Nn \g_@@_preamble_tl
1568   { ! { \skip_horizontal:N 2\l_@@_radius_dim } }

```

The command `\@@_vdottedline:n` is protected, and, therefore, won't be expanded before writing on `\g_@@_internal_code_after_tl`.

```

1569   \tl_gput_right:Nx \g_@@_internal_code_after_tl
1570   { \@@_vdottedline:n { \int_use:N \c@jCol } }
1571   \@@_patch_preamble:n
1572 }

```

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.

```

1573 \cs_new_protected:Npn \@@_patch_preamble_viii:n #1
1574 {
1575   \str_if_eq:nnTF { #1 } { < }
1576   \@@_patch_preamble_ix:n
1577   {
1578     \bool_if:NT \l_@@_vlines_bool
1579     {
1580       \tl_gput_right:Nn \g_@@_preamble_tl
1581       { ! { \skip_horizontal:N \arrayrulewidth } }
1582     }
1583     \@@_patch_preamble:n { #1 }
1584   }
1585 }
1586 \cs_new_protected:Npn \@@_patch_preamble_ix:n #1
1587 {
1588   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
1589   \@@_patch_preamble_viii:n
1590 }

```

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`).

```

1591 \cs_new_protected:Npn \@@_put_box_in_flow:
1592 {
1593   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
1594   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
1595   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
1596   { \box_use_drop:N \l_tmpa_box }
1597   \@@_put_box_in_flow_i:
1598 }

```

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).

```

1599 \cs_new_protected:Npn \@@_put_box_in_flow_i:
1600 {
1601   \pgfpicture
1602   \@@_qpoint:n { row - 1 }
1603   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1604   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
1605   \dim_gadd:Nn \g_tmpa_dim \pgf@y
1606   \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).

```

1607   \str_if_in:NnTF \l_@@_baseline_tl { line- }
1608   {
1609     \int_set:Nn \l_tmpa_int
1610     {
1611       \str_range:Nnn
1612       \l_@@_baseline_tl
1613       6
1614       { \tl_count:V \l_@@_baseline_tl }
1615     }
1616     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
1617   }
1618   {
1619     \str_case:VnF \l_@@_baseline_tl
1620     {
1621       { t } { \int_set:Nn \l_tmpa_int 1 }
1622       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
1623     }
1624     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
1625     \bool_lazy_or:nnT
1626     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
1627     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
1628     {
1629       \@@_error:n { bad~value~for~baseline }
1630       \int_set:Nn \l_tmpa_int 1
1631     }
1632     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

1633   \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
1634   }
1635   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

1636   \endpgfpicture
1637   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
1638   \box_use_drop:N \l_tmpa_box
1639   }

```

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).

```

1640   \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
1641   {

```

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

```

1642   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
1643   \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.

```

1644   \@@_create_extra_nodes:
1645   \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
1646   \bool_lazy_or:nnT
1647   { \int_compare_p:nNn \c@tabularnote > 0 }
1648   { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
1649   \@@_insert_tabularnotes:
1650   \end { minipage }
1651   }

```

```

1652 \cs_new_protected:Npn \@@_insert_tabularnotes:
1653 {
1654   \skip_vertical:N 0.65ex

```

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

```

1655   \group_begin:
1656   \l_@@_notes_code_before_tl
1657   \tl_if_empty:NF \l_@@_tabularnote_tl { \l_@@_tabularnote_tl \par }

```

We compose the tabular notes with a list of `enumitem`. The `\strut` and the `\unskip` are designed to give the ability to put a `\bottomrule` at the end of the notes with a good vertical space.

```

1658   \int_compare:nNnT \c@tabularnote > 0
1659   {
1660     \bool_if:NTF \l_@@_notes_para_bool
1661     {
1662       \begin { tabularnotes* }
1663       \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
1664       \end { tabularnotes* }

```

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

```

1665       \par
1666     }
1667   {
1668     \tabularnotes
1669     \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
1670     \endtabularnotes
1671   }
1672 }
1673 \unskip
1674 \group_end:
1675 \bool_if:NT \l_@@_notes_bottomrule_bool
1676 {
1677   \bool_if:NTF \c_@@_booktabs_loaded_bool
1678   {

```

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

```

1679     \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
1680     { \CT@arc@ \hrule height \heavyrulewidth }
1681   }
1682   { \@@_error:n { bottomule~without~booktabs } }
1683 }
1684 \l_@@_notes_code_after_tl
1685 \seq_gclear:N \g_@@_tabularnotes_seq
1686 \int_gzero:N \c@tabularnote
1687 }

```

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`.

```

1688 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
1689 {
1690   \pgfpicture
1691     \@@_qpoint:n { row - 1 }
1692     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1693     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
1694     \dim_gsub:Nn \g_tmpa_dim \pgf@y
1695   \endpgfpicture
1696   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
1697   \int_compare:nNnT \l_@@_first_row_int = 0
1698   {
1699     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
1700     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim

```

```

1701     }
1702     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
1703 }

```

Now, the general case.

```

1704 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
1705 {

```

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

```

1706     \tl_if_eq:NnT \l_@@_baseline_tl { t }
1707     { \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`.

```

1708     \pgfpicture
1709     \@@_qpoint:n { row - 1 }
1710     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1711     \str_if_in:NnTF \l_@@_baseline_tl { line- }
1712     {
1713         \int_set:Nn \l_tmpa_int
1714         {
1715             \str_range:Nnn
1716             \l_@@_baseline_tl
1717             6
1718             { \tl_count:V \l_@@_baseline_tl }
1719         }
1720         \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
1721     }
1722     {
1723         \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
1724         \bool_lazy_or:nnT
1725         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
1726         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
1727         {
1728             \@@_error:n { bad-value-for-baseline }
1729             \int_set:Nn \l_tmpa_int 1
1730         }
1731         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
1732     }
1733     \dim_gsub:Nn \g_tmpa_dim \pgf@y
1734     \endpgfpicture
1735     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
1736     \int_compare:nNnT \l_@@_first_row_int = 0
1737     {
1738         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
1739         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
1740     }
1741     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
1742 }

```

The command `\@@_put_box_in_flow_bis:` is used when the option `max-delimiter-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.

```

1743 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
1744 {

```

We will compute the real width of both delimiters used.

```

1745     \dim_zero_new:N \l_@@_real_left_delim_dim
1746     \dim_zero_new:N \l_@@_real_right_delim_dim
1747     \hbox_set:Nn \l_tmpb_box
1748     {
1749         \c_math_toggle_token
1750         \left #1
1751         \vcenter

```



```

1752     {
1753         \vbox_to_ht:nn
1754         { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
1755         { }
1756     }
1757     \right .
1758     \c_math_toggle_token
1759 }
1760 \dim_set:Nn \l_@@_real_left_delim_dim
1761 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
1762 \hbox_set:Nn \l_tmpb_box
1763 {
1764     \c_math_toggle_token
1765     \left .
1766     \vbox_to_ht:nn
1767     { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
1768     { }
1769     \right #2
1770     \c_math_toggle_token
1771 }
1772 \dim_set:Nn \l_@@_real_right_delim_dim
1773 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

1774     \skip_horizontal:N \l_@@_left_delim_dim
1775     \skip_horizontal:N -\l_@@_real_left_delim_dim
1776     \@@_put_box_in_flow:
1777     \skip_horizontal:N \l_@@_right_delim_dim
1778     \skip_horizontal:N -\l_@@_real_right_delim_dim
1779 }

```

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).

```

1780 \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, a standard error will be raised by LaTeX for incorrect nested environments).

```

1781 {
1782     \peek_meaning_ignore_spaces:NTF \end \@@_analyze_end:Nn

```

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

```

1783     { \exp_args:NV \@@_array: \g_@@_preamble_tl }
1784 }
1785 {
1786     \@@_create_col_nodes:
1787     \endarray
1788 }

```

When the key `light-syntax` is in force, we use an environment which takes its whole body as an argument (with the specifier `b` of `xparse`).

```

1789 \NewDocumentEnvironment { @@-light-syntax } { b }
1790 {

```

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`.

```

1791     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }

```

```

1792 \tl_map_inline:nn { #1 }
1793 {
1794   \str_if_eq:nnT { ##1 } { & }
1795   { \@@_fatal:n { ampersand-in~light-syntax } }
1796   \str_if_eq:nnT { ##1 } { \ }
1797   { \@@_fatal:n { double-backslash-in~light-syntax } }
1798 }

```

Now, you extract the `code-after` 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`.

```

1799 \@@_light_syntax_i #1 \CodeAfter \q_stop
1800 }

```

Now, the second part of the environment. It is empty. That's not surprising because we have caught the whole body of the environment with the specifier `b` provided by `xparse`.

```

1801 { }
1802 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
1803 {
1804   \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).

```

1805 \seq_gclear_new:N \g_@@_rows_seq
1806 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
1807 \exp_args:NNV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to know that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```

1808 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1809 { \int_set:Nn \l_@@_last_row_int { \seq_count:N \g_@@_rows_seq } }

```

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

```

1810 \exp_args:NV \@@_array: \g_@@_preamble_tl

```

We need a global affectation because, when executing `\l_tmpa_tl`, we will exit the first cell of the array.

```

1811 \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
1812 \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
1813 \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
1814 \@@_create_col_nodes:
1815 \endarray
1816 }
1817 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
1818 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
1819 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
1820 {
1821   \seq_gclear_new:N \g_@@_cells_seq
1822   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
1823   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
1824   \l_tmpa_tl
1825   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
1826 }

```

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).

```

1827 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
1828 {
1829   \str_if_eq:VnT \g_@@_name_env_str { #2 }
1830   { \@@_fatal:n { empty~environment } }

```

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

```
1831     \end { #2 }
1832 }
```

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).

```
1833 \cs_new:Npn \@@_create_col_nodes:
1834 {
1835     \crrc
1836     \int_compare:nNnT \l_@@_first_col_int = 0
1837     {
1838         \omit
1839         \hbox_overlap_left:n
1840         {
1841             \bool_if:NT \l_@@_code_before_bool
1842             { \pgfsys@markposition { \@@_env: - col - 0 } }
1843             \pgfpicture
1844             \pgfrememberpicturepositiononpagetrue
1845             \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
1846             \str_if_empty:NF \l_@@_name_str
1847             { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
1848             \endpgfpicture
1849             \skip_horizontal:N 2\col@sep
1850             \skip_horizontal:N \g_@@_width_first_col_dim
1851         }
1852         &
1853     }
1854     \omit
```

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

```
1855     \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`).

```
1856     \int_compare:nNnTF \l_@@_first_col_int = 0
1857     {
1858         \bool_if:NT \l_@@_code_before_bool
1859         {
1860             \hbox
1861             {
1862                 \skip_horizontal:N -0.5\arrayrulewidth
1863                 \pgfsys@markposition { \@@_env: - col - 1 }
1864                 \skip_horizontal:N 0.5\arrayrulewidth
1865             }
1866         }
1867         \pgfpicture
1868         \pgfrememberpicturepositiononpagetrue
1869         \pgfcoordinate { \@@_env: - col - 1 }
1870         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
1871         \str_if_empty:NF \l_@@_name_str
1872         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
1873         \endpgfpicture
1874     }
1875     {
1876         \bool_if:NT \l_@@_code_before_bool
1877         {
1878             \hbox
1879             {
1880                 \skip_horizontal:N 0.5\arrayrulewidth
1881                 \pgfsys@markposition { \@@_env: - col - 1 }
1882                 \skip_horizontal:N -0.5\arrayrulewidth
```

```

1883     }
1884   }
1885   \pgfpicture
1886   \pgfrememberpicturepositiononpagetrue
1887   \pgfcoordinate { \@@_env: - col - 1 }
1888   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
1889   \str_if_empty:NF \l_@@_name_str
1890   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
1891   \endpgfpicture
1892 }

```

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.

```

1893   \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
1894   \bool_if:NF \l_@@_auto_columns_width_bool
1895   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
1896   {
1897     \bool_lazy_and:nnTF
1898     \l_@@_auto_columns_width_bool
1899     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
1900     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
1901     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
1902     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
1903   }
1904   \skip_horizontal:N \g_tmpa_skip
1905   \hbox
1906   {
1907     \bool_if:NT \l_@@_code_before_bool
1908     {
1909       \hbox
1910       {
1911         \skip_horizontal:N -0.5\arrayrulewidth
1912         \pgfsys@markposition { \@@_env: - col - 2 }
1913         \skip_horizontal:N 0.5\arrayrulewidth
1914       }
1915     }
1916     \pgfpicture
1917     \pgfrememberpicturepositiononpagetrue
1918     \pgfcoordinate { \@@_env: - col - 2 }
1919     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
1920     \str_if_empty:NF \l_@@_name_str
1921     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
1922     \endpgfpicture
1923   }

```

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.

```

1924   \int_gset:Nn \g_tmpa_int 1
1925   \bool_if:NTF \g_@@_last_col_found_bool
1926   { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
1927   { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
1928   {
1929     &
1930     \omit
1931     \int_gincr:N \g_tmpa_int

```

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

```

1932     \skip_horizontal:N \g_tmpa_skip
1933     \bool_if:NT \l_@@_code_before_bool
1934     {

```

```

1935         \hbox
1936         {
1937             \skip_horizontal:N -0.5\arrayrulewidth
1938             \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1939             \skip_horizontal:N 0.5\arrayrulewidth
1940         }
1941     }

```

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

```

1942     \pgfpicture
1943     \pgfrememberpicturepositiononpagetrue
1944     \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1945     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
1946     \str_if_empty:NF \l_@@_name_str
1947     {
1948         \pgfnodealias
1949         { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
1950         { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1951     }
1952     \endpgfpicture
1953 }
1954 \bool_if:NT \g_@@_last_col_found_bool
1955 {
1956     \hbox_overlap_right:n
1957     {
1958         % \skip_horizontal:N \col@sep
1959         \skip_horizontal:N \g_@@_width_last_col_dim
1960         \bool_if:NT \l_@@_code_before_bool
1961         {
1962             \pgfsys@markposition
1963             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1964         }
1965         \pgfpicture
1966         \pgfrememberpicturepositiononpagetrue
1967         \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1968         \pgfpointorigin
1969         \str_if_empty:NF \l_@@_name_str
1970         {
1971             \pgfnodealias
1972             { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
1973             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1974         }
1975         \endpgfpicture
1976     }
1977 }
1978 \cr
1979 }

```

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

```

1980 \tl_const:Nn \c_@@_preamble_first_col_tl
1981 {
1982     >
1983     {
1984         \bool_gset_true:N \g_@@_after_col_zero_bool
1985         \@@_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.

```

1986         \hbox_set:Nw \l_@@_cell_box
1987         \@@_math_toggle_token:
1988         \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”.

```

1989     \bool_lazy_and:nnT
1990     { \int_compare_p:nNn \c@iRow > 0 }
1991     {
1992         \bool_lazy_or_p:nn
1993         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
1994         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
1995     }
1996     {
1997         \l_@@_code_for_first_col_tl
1998         \xglobal \colorlet { nicematrix-first-col } { . }
1999     }
2000 }

```

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`.

```

2001     l
2002     <
2003     {
2004         \@@_math_toggle_token:
2005         \hbox_set_end:
2006         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2007         \@@_adjust_size_box:
2008         \@@_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.

```

2009     \dim_gset:Nn \g_@@_width_first_col_dim
2010     { \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.

```

2011     \hbox_overlap_left:n
2012     {
2013         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2014         \@@_node_for_the_cell:
2015         { \box_use_drop:N \l_@@_cell_box }
2016         \skip_horizontal:N \l_@@_left_delim_dim
2017         \skip_horizontal:N \l_@@_left_margin_dim
2018         \skip_horizontal:N \l_@@_extra_left_margin_dim
2019     }
2020     \bool_gset_false:N \g_@@_empty_cell_bool
2021     \skip_horizontal:N -2\col@sep
2022 }
2023 }

```

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

```

2024 \tl_const:Nn \c_@@_preamble_last_col_tl
2025 {
2026     >
2027     {

```

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

```

2028     \bool_gset_true:N \g_@@_last_col_found_bool
2029     \int_gincr:N \c@jCol
2030     \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.

```

2031     \hbox_set:Nw \l_@@_cell_box
2032     \@@_math_toggle_token:
2033     \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”.

```

2034     \int_compare:nNnT \c@iRow > 0
2035     {
2036         \bool_lazy_or:nnT
2037         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2038         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2039         {
2040             \l_@@_code_for_last_col_tl
2041             \xglobal \colorlet { nicematrix-last-col } { . }
2042         }
2043     }
2044 }
2045 1
2046 <
2047 {
2048     \@@_math_toggle_token:
2049     \hbox_set_end:
2050     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2051     \@@_adjust_size_box:
2052     \@@_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.

```

2053     \dim_gset:Nn \g_@@_width_last_col_dim
2054     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2055     \skip_horizontal:N -2\col@sep

```

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

```

2056     \hbox_overlap_right:n
2057     {
2058         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2059         {
2060             \skip_horizontal:N \l_@@_right_delim_dim
2061             \skip_horizontal:N \l_@@_right_margin_dim
2062             \skip_horizontal:N \l_@@_extra_right_margin_dim
2063             \@@_node_for_the_cell:
2064         }
2065     }
2066     \bool_gset_false:N \g_@@_empty_cell_bool
2067 }
2068 }

```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}` but, in fact, there is a flag `\l_@@_NiceArray_bool`. In `{NiceArrayWithDelims}`, some special code will be executed if this flag is raised.

```

2069 \NewDocumentEnvironment { NiceArray } { }
2070 {
2071     \bool_set_true:N \l_@@_NiceArray_bool
2072     \str_if_empty:NT \g_@@_name_env_str
2073     { \str_gset:Nn \g_@@_name_env_str { NiceArray } }

```

We put `.` and `.` for the delimiters but, in fact, that doesn't matter because these arguments won't be used in `{NiceArrayWithDelims}` (because the flag `\l_@@_NiceArray_bool` is raised).

```

2074     \NiceArrayWithDelims . .
2075 }
2076 { \endNiceArrayWithDelims }

```

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

```

2077 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2078 {

```

```

2079 \NewDocumentEnvironment { #1 NiceArray } { }
2080 {
2081   \str_if_empty:NT \g_@@_name_env_str
2082   { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2083   \@@_test_if_math_mode:
2084   \NiceArrayWithDelims #2 #3
2085 }
2086 { \endNiceArrayWithDelims }
2087 }
2088 \@@_def_env:nnn p ( )
2089 \@@_def_env:nnn b [ ]
2090 \@@_def_env:nnn B \{ \}
2091 \@@_def_env:nnn v | |
2092 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2093 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2094 {
2095   \bool_set_true:N \l_@@_Matrix_bool
2096   \use:c { #1 NiceArray }
2097   {
2098     *
2099     {
2100       \int_compare:nNnTF \l_@@_last_col_int < 0
2101       \c@MaxMatrixCols
2102       { \@@_pred:n \l_@@_last_col_int }
2103     }
2104     { > \@@_Cell: #2 < \@@_end_Cell: }
2105   }
2106 }
2107 \clist_map_inline:nn { { } , p , b , B , v , V }
2108 {
2109   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2110   {
2111     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2112     \tl_set:Nn \l_@@_type_of_col_tl c
2113     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2114     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2115   }
2116   { \use:c { end #1 NiceArray } }
2117 }

```

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

```

2118 \cs_new_protected:Npn \@@_NotEmpty:
2119 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

The environments {NiceTabular} and {NiceTabular*}

```

2120 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2121 {
2122   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2123   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2124   \bool_set_true:N \l_@@_NiceTabular_bool
2125   \NiceArray { #2 }
2126 }
2127 { \endNiceArray }
2128 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }

```



```

2129 {
2130   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2131   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
2132   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2133   \bool_set_true:N \l_@@_NiceTabular_bool
2134   \NiceArray { #3 }
2135 }
2136 { \endNiceArray }

```

After the construction of the array

```

2137 \cs_new_protected:Npn \@@_after_array:
2138 {
2139   \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.

```

2140   \bool_if:NT \g_@@_last_col_found_bool
2141   { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }

```

If we are in an environment without preamble (like `{NiceMatrix}` or `{pNiceMatrix}`) and if the option `last-col` has been used without value we fix the real value of `\l_@@_last_col_int`.

```

2142   \bool_if:NT \l_@@_last_col_without_value_bool
2143   {
2144     \dim_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int
2145     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2146     \iow_shipout:Nx \@mainaux
2147     {
2148       \cs_gset:cpn { @@_last_col_ \int_use:N \g_@@_env_int }
2149       { \int_use:N \g_@@_col_total_int }
2150     }
2151     \str_if_empty:NF \l_@@_name_str
2152     {
2153       \iow_shipout:Nx \@mainaux
2154       {
2155         \cs_gset:cpn { @@_last_col_ \l_@@_name_str }
2156         { \int_use:N \g_@@_col_total_int }
2157       }
2158     }
2159     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2160   }

```

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

```

2161   \bool_if:NT \l_@@_last_row_without_value_bool
2162   {
2163     \dim_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int

```

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

```

2164   \bool_if:NF \l_@@_light_syntax_bool
2165   {
2166     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2167     \iow_shipout:Nx \@mainaux
2168     {
2169       \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int }
2170       { \int_use:N \g_@@_row_total_int }
2171     }

```

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

```

2172     \str_if_empty:NF \l_@@_name_str
2173     {
2174         \iow_shipout:Nx \@mainaux
2175         {
2176             \cs_gset:cpn { @@_last_row_ \l_@@_name_str }
2177             { \int_use:N \g_@@_row_total_int }
2178         }
2179     }
2180     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2181 }
2182

```

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

```

2183     \bool_if:NT \l_@@_code_before_bool
2184     {
2185         \iow_now:Nn \@mainaux \ExplSyntaxOn
2186         \iow_now:Nx \@mainaux
2187         { \seq_clear_new:c { @@_size _ \int_use:N \g_@@_env_int _ seq } }
2188         \iow_now:Nx \@mainaux
2189         {
2190             \seq_gset_from_clist:cn { @@_size _ \int_use:N \g_@@_env_int _ seq }
2191             {
2192                 \int_use:N \l_@@_first_row_int ,
2193                 \int_use:N \g_@@_row_total_int ,
2194                 \int_use:N \l_@@_first_col_int ,

```

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

```

2195         \bool_lazy_and:nnTF
2196         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
2197         { \bool_not_p:n \g_@@_last_col_found_bool }
2198         \@@_succ:n
2199         \int_use:N
2200         \g_@@_col_total_int
2201     }

```

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

```

2202         \seq_gset_from_clist:cn
2203         { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
2204         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
2205     }
2206     \iow_now:Nn \@mainaux \ExplSyntaxOff
2207 }

```

By default, the diagonal lines will be parallelized⁴⁵. 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.

```

2208     \bool_if:NT \l_@@_parallelize_diags_bool
2209     {
2210         \int_gzero_new:N \g_@@_ddots_int
2211         \int_gzero_new:N \g_@@_iddots_int

```

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

```

2212         \dim_gzero_new:N \g_@@_delta_x_one_dim
2213         \dim_gzero_new:N \g_@@_delta_y_one_dim
2214         \dim_gzero_new:N \g_@@_delta_x_two_dim
2215         \dim_gzero_new:N \g_@@_delta_y_two_dim
2216     }
2217     \int_zero_new:N \l_@@_initial_i_int

```

⁴⁵It's possible to use the option parallelize-diags to disable this parallelization.

```

2218 \int_zero_new:N \l_@@_initial_j_int
2219 \int_zero_new:N \l_@@_final_i_int
2220 \int_zero_new:N \l_@@_final_j_int
2221 \bool_set_false:N \l_@@_initial_open_bool
2222 \bool_set_false:N \l_@@_final_open_bool

```

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

```

2223 \bool_if:NT \l_@@_small_bool
2224 {
2225     \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
2226     \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }

```

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

```

2227     \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
2228 }

```

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

```

2229 \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it will do nothing.

```

2230 \@@_compute_corners:

```

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

```

2231 \@@_adjust_pos_of_blocks_seq:

```

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

```

2232 \bool_lazy_all:nT
2233 {
2234     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
2235     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
2236     { \seq_if_empty_p:N \l_@@_empty_corner_cells_seq }
2237 }
2238 {
2239     \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn
2240     \cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn
2241 }
2242 \bool_if:NT \l_@@_hlines_bool \@@_draw_hlines:
2243 \bool_if:NT \l_@@_vlines_bool \@@_draw_vlines:
2244 \g_@@_internal_code_after_tl
2245 \tl_gclear:N \g_@@_internal_code_after_tl

```

Now, the code-after.

```

2246 \bool_if:NT \c_@@_tikz_loaded_bool
2247 {
2248     \tikzset
2249     {
2250         every~picture / .style =
2251         {
2252             overlay ,
2253             remember~picture ,
2254             name~prefix = \@@_env: -
2255         }

```

```

2256     }
2257   }
2258   \cs_set_eq:NN \line \@@_line

```

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.

```

2259   \cs_set_eq:NN \CodeAfter \prg_do_nothing:

```

And here's the code-after:

```

2260   \g_nicematrix_code_after_tl
2261   \tl_gclear:N \g_nicematrix_code_after_tl
2262   \group_end:

```

`\g_nicematrix_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor` (when the key `colortbl-like` is in force). These instructions will be written on the `aux` file to be added to the code-before in the next run.

```

2263   \tl_if_empty:NF \g_nicematrix_code_before_tl
2264   {

```

The command `\rowcolor` in `tabular` will in fact use `\rectanglecolor` in order to follow the behaviour of `\rowcolor` of `colortbl`. That's why there may be a command `\rectanglecolor` in `\g_nicematrix_code_before_tl`. In order to avoid an error during the expansion, we define a protected version of `\rectanglecolor`.

```

2265     \cs_set_protected:Npn \rectanglecolor { }
2266     \cs_set_protected:Npn \columncolor { }
2267     \iow_now:Nn \@mainaux \ExplSyntaxOn
2268     \iow_now:Nx \@mainaux
2269     {
2270       \tl_gset:cn
2271       { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
2272       { \exp_not:V \g_nicematrix_code_before_tl }
2273     }
2274     \iow_now:Nn \@mainaux \ExplSyntaxOff
2275     \bool_set_true:N \l_@@_code_before_bool
2276   }

2277   \str_gclear:N \g_@@_name_env_str
2278   \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁴⁶. 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`.

```

2279   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
2280   }

```

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).

```

2281   \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
2282   {
2283     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
2284     { \@@_adjust_pos_of_blocks_seq_i:nnnn ##1 }
2285   }

```

The following command must *not* be protected.

```

2286   \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnn #1 #2 #3 #4
2287   {

```

⁴⁶e.g. `\color[rgb]{0.5,0.5,0}`

```

2288 { #1 }
2289 { #2 }
2290 {
2291   \int_compare:nNnTF { #3 } > { 99 }
2292   { \int_use:N \c@iRow }
2293   { #3 }
2294 }
2295 {
2296   \int_compare:nNnTF { #4 } > { 99 }
2297   { \int_use:N \c@jCol }
2298   { #4 }
2299 }
2300 }

```

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).

```

2301 \AtBeginDocument
2302 {
2303   \cs_new_protected:Npx \@@_draw_dotted_lines:
2304   {
2305     \c_@@_pgfortikzpicture_tl
2306     \@@_draw_dotted_lines_i:
2307     \c_@@_endpgfortikzpicture_tl
2308   }
2309 }

```

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

```

2310 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
2311 {
2312   \pgfrememberpicturerepositiononpagetrue
2313   \pgf@relevantforpicturesizefalse
2314   \g_@@_HVdotsfor_lines_tl
2315   \g_@@_Vdots_lines_tl
2316   \g_@@_Ddots_lines_tl
2317   \g_@@_Iddots_lines_tl
2318   \g_@@_Cdots_lines_tl
2319   \g_@@_Ldots_lines_tl
2320 }

2321 \cs_new_protected:Npn \@@_restore_iRow_jCol:
2322 {
2323   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
2324   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
2325 }

```

We draw the dotted lines

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

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

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

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;

- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

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

```
2326 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
2327 {
```

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

```
2328 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
2329 \int_set:Nn \l_@@_initial_i_int { #1 }
2330 \int_set:Nn \l_@@_initial_j_int { #2 }
2331 \int_set:Nn \l_@@_final_i_int { #1 }
2332 \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.

```
2333 \bool_set_false:N \l_@@_stop_loop_bool
2334 \bool_do_until:Nn \l_@@_stop_loop_bool
2335 {
2336   \int_add:Nn \l_@@_final_i_int { #3 }
2337   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
2338 \bool_set_false:N \l_@@_final_open_bool
2339 \int_compare:nNnTF \l_@@_final_i_int > \c@iRow
2340 {
2341   \int_compare:nNnTF { #3 } = 1
2342   { \bool_set_true:N \l_@@_final_open_bool }
2343   {
2344     \int_compare:nNnT \l_@@_final_j_int > \c@jCol
2345     { \bool_set_true:N \l_@@_final_open_bool }
2346   }
2347 }
2348 {
2349   \int_compare:nNnTF \l_@@_final_j_int < 1
2350   {
2351     \int_compare:nNnT { #4 } = { -1 }
2352     { \bool_set_true:N \l_@@_final_open_bool }
2353   }
2354   {
2355     \int_compare:nNnT \l_@@_final_j_int > \c@jCol
2356     {
2357       \int_compare:nNnT { #4 } = 1
2358       { \bool_set_true:N \l_@@_final_open_bool }
2359     }
2360   }
2361 }
2362 \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.

```
2363 {
```

We do a step backwards.

```

2364     \int_sub:Nn \l_@@_final_i_int { #3 }
2365     \int_sub:Nn \l_@@_final_j_int { #4 }
2366     \bool_set_true:N \l_@@_stop_loop_bool
2367 }

```

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`.

```

2368 {
2369   \cs_if_exist:cTF
2370   {
2371     @@ _ dotted _
2372     \int_use:N \l_@@_final_i_int -
2373     \int_use:N \l_@@_final_j_int
2374   }
2375   {
2376     \int_sub:Nn \l_@@_final_i_int { #3 }
2377     \int_sub:Nn \l_@@_final_j_int { #4 }
2378     \bool_set_true:N \l_@@_final_open_bool
2379     \bool_set_true:N \l_@@_stop_loop_bool
2380   }
2381   {
2382     \cs_if_exist:cTF
2383     {
2384       pgf @ sh @ ns @ \@@_env:
2385       - \int_use:N \l_@@_final_i_int
2386       - \int_use:N \l_@@_final_j_int
2387     }
2388     { \bool_set_true:N \l_@@_stop_loop_bool }
2389   }
2390 }

```

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 environnement), 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.

```

2389 {
2390   \cs_set:cpn
2391   {
2392     @@ _ dotted _
2393     \int_use:N \l_@@_final_i_int -
2394     \int_use:N \l_@@_final_j_int
2395   }
2396   { }
2397 }
2398 }
2399 }
2400 }

```

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

```

2401 \bool_set_false:N \l_@@_stop_loop_bool
2402 \bool_do_until:Nn \l_@@_stop_loop_bool
2403 {
2404   \int_sub:Nn \l_@@_initial_i_int { #3 }
2405   \int_sub:Nn \l_@@_initial_j_int { #4 }
2406   \bool_set_false:N \l_@@_initial_open_bool
2407   \int_compare:nNnTF \l_@@_initial_i_int < 1
2408   {
2409     \int_compare:nNnTF { #3 } = 1
2410     { \bool_set_true:N \l_@@_initial_open_bool }
2411     {

```

```

2412         \int_compare:nNt \l_@@_initial_j_int = 0
2413         { \bool_set_true:N \l_@@_initial_open_bool }
2414     }
2415 }
2416 {
2417     \int_compare:nNtF \l_@@_initial_j_int < 1
2418     {
2419         \int_compare:nNt { #4 } = 1
2420         { \bool_set_true:N \l_@@_initial_open_bool }
2421     }
2422     {
2423         \int_compare:nNt \l_@@_initial_j_int > \c@jCol
2424         {
2425             \int_compare:nNt { #4 } = { -1 }
2426             { \bool_set_true:N \l_@@_initial_open_bool }
2427         }
2428     }
2429 }
2430 \bool_if:NtF \l_@@_initial_open_bool
2431 {
2432     \int_add:Nn \l_@@_initial_i_int { #3 }
2433     \int_add:Nn \l_@@_initial_j_int { #4 }
2434     \bool_set_true:N \l_@@_stop_loop_bool
2435 }
2436 {
2437     \cs_if_exist:cTF
2438     {
2439         @@ _ dotted _
2440         \int_use:N \l_@@_initial_i_int -
2441         \int_use:N \l_@@_initial_j_int
2442     }
2443     {
2444         \int_add:Nn \l_@@_initial_i_int { #3 }
2445         \int_add:Nn \l_@@_initial_j_int { #4 }
2446         \bool_set_true:N \l_@@_initial_open_bool
2447         \bool_set_true:N \l_@@_stop_loop_bool
2448     }
2449     {
2450         \cs_if_exist:cTF
2451         {
2452             pgf @ sh @ ns @ \@@_env:
2453             - \int_use:N \l_@@_initial_i_int
2454             - \int_use:N \l_@@_initial_j_int
2455         }
2456         { \bool_set_true:N \l_@@_stop_loop_bool }
2457         {
2458             \cs_set:cpn
2459             {
2460                 @@ _ dotted _
2461                 \int_use:N \l_@@_initial_i_int -
2462                 \int_use:N \l_@@_initial_j_int
2463             }
2464             { }
2465         }
2466     }
2467 }
2468 }

```

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.

```

2469 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
2470 {
2471     { \int_use:N \l_@@_initial_i_int }
2472     { \int_use:N \l_@@_initial_j_int }

```



```

2473     { \int_use:N \l_@@_final_i_int }
2474     { \int_use:N \l_@@_final_j_int }
2475   }
2476 }

2477 \cs_new_protected:Npn \@@_set_initial_coords:
2478 {
2479   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2480   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2481 }
2482 \cs_new_protected:Npn \@@_set_final_coords:
2483 {
2484   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2485   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2486 }
2487 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
2488 {
2489   \pgfpointanchor
2490   {
2491     \@@_env:
2492     - \int_use:N \l_@@_initial_i_int
2493     - \int_use:N \l_@@_initial_j_int
2494   }
2495   { #1 }
2496   \@@_set_initial_coords:
2497 }
2498 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
2499 {
2500   \pgfpointanchor
2501   {
2502     \@@_env:
2503     - \int_use:N \l_@@_final_i_int
2504     - \int_use:N \l_@@_final_j_int
2505   }
2506   { #1 }
2507   \@@_set_final_coords:
2508 }

```

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.

```

2509 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
2510 {
2511   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2512   {
2513     \@@_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.

```

2514   \group_begin:
2515     \int_compare:nNnTF { #1 } = 0
2516     { \color { nicematrix-first-row } }
2517     {

```

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.

```

2518       \int_compare:nNnT { #1 } = \l_@@_last_row_int
2519       { \color { nicematrix-last-row } }
2520     }
2521     \keys_set:nn { NiceMatrix / xdots } { #3 }
2522     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2523     \@@_actually_draw_Ldots:
2524   \group_end:
2525 }
2526 }

```

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`.

```

2527 \cs_new_protected:Npn \@@_actually_draw_Ldots:
2528 {
2529   \bool_if:NTF \l_@@_initial_open_bool
2530   {
2531     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2532     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2533     \dim_add:Nn \l_@@_x_initial_dim \col@sep
2534     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
2535     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2536   }
2537   { \@@_set_initial_coords_from_anchor:n { base-east } }
2538   \bool_if:NTF \l_@@_final_open_bool
2539   {
2540     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2541     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2542     \dim_sub:Nn \l_@@_x_final_dim \col@sep
2543     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
2544     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2545   }
2546   { \@@_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 (?).

```

2547   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
2548   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
2549   \@@_draw_line:
2550 }

```

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.

```

2551 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
2552 {
2553   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2554   {
2555     \@@_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.

```

2556   \group_begin:
2557   \int_compare:nNnTF { #1 } = 0
2558   { \color { nicematrix-first-row } }
2559   {

```

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.

```

2560     \int_compare:nNnT { #1 } = \l_@@_last_row_int
2561     { \color { nicematrix-last-row } }
2562   }
2563   \keys_set:nn { NiceMatrix / xdots } { #3 }

```

```

2564         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2565         \@@_actually_draw_Cdots:
2566     \group_end:
2567 }
2568 }

```

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`.

```

2569 \cs_new_protected:Npn \@@_actually_draw_Cdots:
2570 {
2571     \bool_if:NTF \l_@@_initial_open_bool
2572     {
2573         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2574         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2575         \dim_add:Nn \l_@@_x_initial_dim \col@sep
2576     }
2577     { \@@_set_initial_coords_from_anchor:n { mid-east } }
2578     \bool_if:NTF \l_@@_final_open_bool
2579     {
2580         \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2581         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2582         \dim_sub:Nn \l_@@_x_final_dim \col@sep
2583     }
2584     { \@@_set_final_coords_from_anchor:n { mid-west } }
2585     \bool_lazy_and:nnTF
2586         \l_@@_initial_open_bool
2587         \l_@@_final_open_bool
2588     {
2589         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2590         \dim_set_eq:NN \l_tmpa_dim \pgf@y
2591         \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
2592         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
2593         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
2594     }
2595     {
2596         \bool_if:NT \l_@@_initial_open_bool
2597         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
2598         \bool_if:NT \l_@@_final_open_bool
2599         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
2600     }
2601     \@@_draw_line:
2602 }

```

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.

```

2603 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
2604 {
2605     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2606     {
2607         \@@_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.

```

2608     \group_begin:

```

```

2609     \int_compare:nNnTF { #2 } = 0
2610     { \color { nicematrix-first-col } }
2611     {
2612         \int_compare:nNnT { #2 } = \l_@@_last_col_int
2613         { \color { nicematrix-last-col } }
2614     }
2615     \keys_set:nn { NiceMatrix / xdots } { #3 }
2616     \tl_if_empty:VF \l_@@_xdots_color_tl
2617     { \color { \l_@@_xdots_color_tl } }
2618     \@@_actually_draw_Vdots:
2619 \group_end:
2620 }
2621 }

```

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`.

```

2622 \cs_new_protected:Npn \@@_actually_draw_Vdots:
2623 {

```

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

```

2624     \bool_set_false:N \l_tmpa_bool
2625     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
2626     {
2627         \@@_set_initial_coords_from_anchor:n { south-west }
2628         \@@_set_final_coords_from_anchor:n { north-west }
2629         \bool_set:Nn \l_tmpa_bool
2630         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
2631     }

```

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

```

2632     \bool_if:NTF \l_@@_initial_open_bool
2633     {
2634         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2635         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2636     }
2637     { \@@_set_initial_coords_from_anchor:n { south } }
2638     \bool_if:NTF \l_@@_final_open_bool
2639     {
2640         \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2641         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2642     }
2643     { \@@_set_final_coords_from_anchor:n { north } }
2644     \bool_if:NTF \l_@@_initial_open_bool
2645     {
2646         \bool_if:NTF \l_@@_final_open_bool
2647         {
2648             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2649             \dim_set_eq:NN \l_tmpa_dim \pgf@x
2650             \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
2651             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
2652             \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.

```

2653         \int_compare:nNnT \l_@@_last_col_int > { -2 }
2654         {
2655             \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
2656             {
2657                 \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
2658                 \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
2659                 \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
2660                 \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
2661             }
2662         }
2663     }
2664     { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
2665 }
2666 {
2667     \bool_if:NTF \l_@@_final_open_bool
2668     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
2669     {

```

Now the case where both extremities are closed. The first conditional tests whether the column is of type `c` (`C` of `{NiceArray}`) or may be considered as if.

```

2670         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
2671         {
2672             \dim_set:Nn \l_@@_x_initial_dim
2673             {
2674                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
2675                 \l_@@_x_initial_dim \l_@@_x_final_dim
2676             }
2677             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
2678         }
2679     }
2680 }
2681 \@@_draw_line:
2682 }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

2683 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
2684 {
2685     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2686     {
2687         \@@_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.

```

2688     \group_begin:
2689     \keys_set:nn { NiceMatrix / xdots } { #3 }
2690     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2691     \@@_actually_draw_Ddots:
2692     \group_end:
2693 }
2694 }

```

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`.

```

2695 \cs_new_protected:Npn \@@_actually_draw_Ddots:
2696 {
2697   \bool_if:NTF \l_@@_initial_open_bool
2698   {
2699     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2700     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2701     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2702     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2703   }
2704   { \@@_set_initial_coords_from_anchor:n { south-east } }
2705   \bool_if:NTF \l_@@_final_open_bool
2706   {
2707     \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2708     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2709     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2710     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2711   }
2712   { \@@_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.

```

2713   \bool_if:NT \l_@@_parallelize_diags_bool
2714   {
2715     \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).

```

2716     \int_compare:nNnTF \g_@@_ddots_int = 1

```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

2717     {
2718       \dim_gset:Nn \g_@@_delta_x_one_dim
2719       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
2720       \dim_gset:Nn \g_@@_delta_y_one_dim
2721       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
2722     }

```

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`.

```

2723     {
2724       \dim_set:Nn \l_@@_y_final_dim
2725       {
2726         \l_@@_y_initial_dim +
2727         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2728         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
2729       }
2730     }
2731   }
2732   \@@_draw_line:
2733 }

```

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.

```

2734 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
2735 {
2736   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2737   {
2738     \@@_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.

```

2739     \group_begin:
2740     \keys_set:nn { NiceMatrix / xdots } { #3 }
2741     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2742     \@@_actually_draw_Iddots:
2743     \group_end:
2744   }
2745 }

```

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`.

```

2746 \cs_new_protected:Npn \@@_actually_draw_Iddots:
2747 {
2748   \bool_if:NTF \l_@@_initial_open_bool
2749   {
2750     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2751     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2752     \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
2753     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2754   }
2755   { \@@_set_initial_coords_from_anchor:n { south-west } }
2756   \bool_if:NTF \l_@@_final_open_bool
2757   {
2758     \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2759     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2760     \@@_qpoint:n { col - \int_use:N \l_@@_final_j_int }
2761     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2762   }
2763   { \@@_set_final_coords_from_anchor:n { north-east } }
2764   \bool_if:NT \l_@@_parallelize_diags_bool
2765   {
2766     \int_gincr:N \g_@@_iddots_int
2767     \int_compare:nNnTF \g_@@_iddots_int = 1
2768     {
2769       \dim_gset:Nn \g_@@_delta_x_two_dim
2770       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
2771       \dim_gset:Nn \g_@@_delta_y_two_dim
2772       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
2773     }
2774     {
2775       \dim_set:Nn \l_@@_y_final_dim
2776       {
2777         \l_@@_y_initial_dim +
2778         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2779         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
2780       }

```

```

2781     }
2782   }
2783   \@@_draw_line:
2784 }

```

The actual instructions for drawing the dotted line with Tikz

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

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

```

2785 \cs_new_protected:Npn \@@_draw_line:
2786 {

```

First, we put the labels.

```

2787   \bool_lazy_and:nnF
2788   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
2789   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
2790   {
2791     \pgfscope
2792     \pgftransformshift
2793     {
2794       \pgfpointlineatime { 0.5 }
2795       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
2796       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
2797     }
2798     \pgftransformrotate
2799     {
2800       \fp_eval:n
2801       {
2802         atand
2803         (
2804           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
2805           \l_@@_x_final_dim - \l_@@_x_initial_dim
2806         )
2807       }
2808     }
2809     \pgfnode
2810     { rectangle }
2811     { south }
2812     {
2813       \c_math_toggle_token
2814       \scriptstyle \l_@@_xdots_up_tl
2815       \c_math_toggle_token
2816     }
2817     { }
2818     { \pgfusepath { } }
2819     \pgfnode
2820     { rectangle }
2821     { north }
2822     {
2823       \c_math_toggle_token
2824       \scriptstyle \l_@@_xdots_down_tl

```



```

2825         \c_math_toggle_token
2826     }
2827     { }
2828     { \pgfusepath { } }
2829 \endpgfscope
2830 }
2831 \pgfrememberpicturepositiononpagetrue
2832 \pgf@relevantforpicturesizefalse
2833 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
2834     \@@_draw_standard_dotted_line:
2835     \@@_draw_non_standard_dotted_line:
2836 }

```

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.

```

2837 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
2838 {
2839     \begin { scope }
2840     \exp_args:No \@@_draw_non_standard_dotted_line:n
2841         { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
2842 }

```

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

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

```

2843 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
2844 {
2845     \draw
2846     [
2847         #1 ,
2848         shorten-> = \l_@@_xdots_shorten_dim ,
2849         shorten-< = \l_@@_xdots_shorten_dim ,
2850     ]
2851         ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
2852         --
2853         ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
2854     \end { scope }
2855 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of points (which give a dotted line with real round points).

```

2856 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
2857 {
2858     \group_begin:

```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of `expl3` to compute this length.

```

2859     \dim_zero_new:N \l_@@_l_dim
2860     \dim_set:Nn \l_@@_l_dim
2861     {
2862         \fp_to_dim:n
2863         {
2864             sqrt
2865             (
2866                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
2867                 +
2868                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
2869             )
2870         }
2871     }

```

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.

```

2872     \bool_lazy_or:nnF
2873     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
2874     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
2875     \@@_draw_standard_dotted_line_i:
2876     \group_end:
2877 }
2878 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
2879 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
2880 {

```

The integer `\l_tmpa_int` is the number of dots of the dotted line.

```

2881     \bool_if:NTF \l_@@_initial_open_bool
2882     {
2883         \bool_if:NTF \l_@@_final_open_bool
2884         {
2885             \int_set:Nn \l_tmpa_int
2886             { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
2887         }
2888         {
2889             \int_set:Nn \l_tmpa_int
2890             {
2891                 \dim_ratio:nn
2892                 { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
2893                 \l_@@_inter_dots_dim
2894             }
2895         }
2896     }
2897     {
2898         \bool_if:NTF \l_@@_final_open_bool
2899         {
2900             \int_set:Nn \l_tmpa_int
2901             {
2902                 \dim_ratio:nn
2903                 { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
2904                 \l_@@_inter_dots_dim
2905             }
2906         }
2907         {
2908             \int_set:Nn \l_tmpa_int
2909             {
2910                 \dim_ratio:nn
2911                 { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
2912                 \l_@@_inter_dots_dim
2913             }
2914         }
2915     }

```

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

```

2916     \dim_set:Nn \l_tmpa_dim
2917     {
2918         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2919         \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
2920     }
2921     \dim_set:Nn \l_tmpb_dim
2922     {
2923         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
2924         \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
2925     }

```

The length ℓ is the length of the dotted line. We note Δ the length between two dots and n the number of intervals between dots. We note $\delta = \frac{1}{2}(\ell - n\Delta)$. The distance between the initial extremity of the line and the first dot will be equal to $k \cdot \delta$ where $k = 0, 1$ or 2 . We first compute this number k in `\l_tmpb_int`.

```

2926   \int_set:Nn \l_tmpb_int
2927   {
2928     \bool_if:NTF \l_@@_initial_open_bool
2929     { \bool_if:NTF \l_@@_final_open_bool 1 0 }
2930     { \bool_if:NTF \l_@@_final_open_bool 2 1 }
2931   }

```

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.

```

2932   \dim_gadd:Nn \l_@@_x_initial_dim
2933   {
2934     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2935     \dim_ratio:nn
2936     { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
2937     { 2 \l_@@_l_dim }
2938     * \l_tmpb_int
2939   }
2940   \dim_gadd:Nn \l_@@_y_initial_dim
2941   {
2942     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
2943     \dim_ratio:nn
2944     { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
2945     { 2 \l_@@_l_dim }
2946     * \l_tmpb_int
2947   }
2948   \pgf@relevantforpicturesizefalse
2949   \int_step_inline:nnn 0 \l_tmpa_int
2950   {
2951     \pgfpathcircle
2952     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
2953     { \l_@@_radius_dim }
2954     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
2955     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
2956   }
2957   \pgfusepathqfill
2958 }

```

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 starred versions of these commands are deprecated since version 3.1 but, as of now, they are still available with an error.

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\AtBeginDocument` and the *arg spec* will be rescanned.

```

2959 \AtBeginDocument
2960 {
2961   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
2962   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

2963 \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
2964 {
2965   \int_compare:nNnTF \c@jCol = 0
2966   { \@@_error:nn { in~first~col } \Ldots }
2967   {
2968     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
2969     { \@@_error:nn { in~last~col } \Ldots }
2970     {
2971       \@@_instruction_of_type:nnn \c_false_bool { Ldots }
2972       { #1 , down = #2 , up = #3 }
2973     }
2974   }
2975   \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ldots }
2976   \bool_gset_true:N \g_@@_empty_cell_bool
2977 }

2978 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
2979 {
2980   \int_compare:nNnTF \c@jCol = 0
2981   { \@@_error:nn { in~first~col } \Cdots }
2982   {
2983     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
2984     { \@@_error:nn { in~last~col } \Cdots }
2985     {
2986       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
2987       { #1 , down = #2 , up = #3 }
2988     }
2989   }
2990   \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_cdots }
2991   \bool_gset_true:N \g_@@_empty_cell_bool
2992 }

2993 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
2994 {
2995   \int_compare:nNnTF \c@iRow = 0
2996   { \@@_error:nn { in~first~row } \Vdots }
2997   {
2998     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
2999     { \@@_error:nn { in~last~row } \Vdots }
3000     {
3001       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
3002       { #1 , down = #2 , up = #3 }
3003     }
3004   }
3005   \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_vdots }
3006   \bool_gset_true:N \g_@@_empty_cell_bool
3007 }

3008 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
3009 {
3010   \int_case:nnF \c@iRow
3011   {
3012     0 { \@@_error:nn { in~first~row } \Ddots }
3013     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
3014   }
3015   {
3016     \int_case:nnF \c@jCol
3017     {
3018       0 { \@@_error:nn { in~first~col } \Ddots }
3019       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
3020     }

```

```

3021         {
3022             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3023             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
3024             { #1 , down = #2 , up = #3 }
3025         }
3026     }
3027 }
3028 \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ddots }
3029 \bool_gset_true:N \g_@@_empty_cell_bool
3030 }

3031 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
3032 {
3033     \int_case:nnF \c@iRow
3034     {
3035         0 { \@@_error:nn { in~first~row } \Iddots }
3036         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
3037     }
3038     {
3039         \int_case:nnF \c@jCol
3040         {
3041             0 { \@@_error:nn { in~first~col } \Iddots }
3042             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
3043         }
3044         {
3045             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3046             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
3047             { #1 , down = #2 , up = #3 }
3048         }
3049     }
3050 }
3051 \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_iddots }
3052 \bool_gset_true:N \g_@@_empty_cell_bool
3053 }
3054 }

```

End of the \AtBeginDocument.

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

```

3055 \keys_define:nn { NiceMatrix / Ddots }
3056 {
3057     draw-first .bool_set:N = \l_@@_draw_first_bool ,
3058     draw-first .default:n = true ,
3059     draw-first .value_forbidden:n = true
3060 }

```

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

```

3061 \cs_new_protected:Npn \@@_Hspace:
3062 {
3063     \bool_gset_true:N \g_@@_empty_cell_bool
3064     \hspace
3065 }

```

In the environment {NiceArray}, the command \multicolumn will be linked to the following command \@@_multicolumn:nnn.

```

3066 \cs_set_eq:NN \@@_old_multicolumn \multicolumn
3067 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3068 {
3069     % \begin{macrocode}
3070     % We have to act in an expandable way since it will begin by a |\multicolumn|.
3071     % \end{macrocode}

```

```

3072 \exp_args:NNe
3073   \@@_old_multicolumn
3074   { #1 }

```

We will have to replace `\tl_lower_case:n` in the future since it seems to be deprecated.

```

3075 {
3076   \exp_args:Ne \str_case:nn { \str_foldcase:n { #2 } }
3077   {
3078     l { > \@@_Cell: l < \@@_end_Cell: }
3079     r { > \@@_Cell: r < \@@_end_Cell: }
3080     c { > \@@_Cell: c < \@@_end_Cell: }
3081     { l | } { > \@@_Cell: l < \@@_end_Cell: | }
3082     { r | } { > \@@_Cell: r < \@@_end_Cell: | }
3083     { c | } { > \@@_Cell: c < \@@_end_Cell: | }
3084     { | l } { | > \@@_Cell: l < \@@_end_Cell: }
3085     { | r } { | > \@@_Cell: r < \@@_end_Cell: }
3086     { | c } { | > \@@_Cell: c < \@@_end_Cell: }
3087     { | l | } { | > \@@_Cell: l < \@@_end_Cell: | }
3088     { | r | } { | > \@@_Cell: r < \@@_end_Cell: | }
3089     { | c | } { | > \@@_Cell: c < \@@_end_Cell: | }
3090   }
3091 }
3092 { #3 }

```

The `\peek_remove_spaces:n` is mandatory.

```

3093 \peek_remove_spaces:n
3094 {
3095   \int_compare:nNnT #1 > 1
3096   {
3097     \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
3098     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3099     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
3100     \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
3101     {
3102       { \int_use:N \c@iRow }
3103       { \int_use:N \c@jCol }
3104       { \int_use:N \c@iRow }
3105       { \int_eval:n { \c@jCol + #1 - 1 } }
3106     }
3107   }
3108   \int_gadd:Nn \c@jCol { #1 - 1 }
3109   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3110   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3111 }
3112 }

```

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`.

```

3113 \cs_new:Npn \@@_Hdotsfor:
3114 {
3115   \bool_lazy_and:nnTF
3116   { \int_compare_p:nNn \c@jCol = 0 }
3117   { \int_compare_p:nNn \l_@@_first_col_int = 0 }
3118   {
3119     \bool_if:NTF \g_@@_after_col_zero_bool
3120     {
3121       \multicolumn { 1 } { c } { }
3122       \@@_Hdotsfor_i
3123     }
3124     { \@@_fatal:n { Hdotsfor~in~col~0 } }
3125   }
3126 }

```

```

3127     \multicolumn { 1 } { c } { }
3128     \@@_Hdotsfor_i
3129 }
3130 }

```

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:`).

```

3131 \AtBeginDocument
3132 {
3133     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3134     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put ! before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

3135     \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
3136     {
3137         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
3138         {
3139             \@@_Hdotsfor:nnnn
3140             { \int_use:N \c@iRow }
3141             { \int_use:N \c@jCol }
3142             { #2 }
3143             {
3144                 #1 , #3 ,
3145                 down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3146             }
3147         }
3148         \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
3149     }
3150 }

```

Enf of `\AtBeginDocument`.

```

3151 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
3152 {
3153     \bool_set_false:N \l_@@_initial_open_bool
3154     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

3155     \int_set:Nn \l_@@_initial_i_int { #1 }
3156     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

3157     \int_compare:nNnTF #2 = 1
3158     {
3159         \int_set:Nn \l_@@_initial_j_int 1
3160         \bool_set_true:N \l_@@_initial_open_bool
3161     }
3162     {
3163         \cs_if_exist:cTF
3164         {
3165             pgf @ sh @ ns @ \@@_env:
3166             - \int_use:N \l_@@_initial_i_int
3167             - \int_eval:n { #2 - 1 }
3168         }
3169         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
3170         {
3171             \int_set:Nn \l_@@_initial_j_int { #2 }
3172             \bool_set_true:N \l_@@_initial_open_bool
3173         }
3174     }
3175     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
3176     {
3177         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }

```

```

3178     \bool_set_true:N \l_@@_final_open_bool
3179   }
3180   {
3181     \cs_if_exist:cTF
3182     {
3183       pgf @ sh @ ns @ \@@_env:
3184       - \int_use:N \l_@@_final_i_int
3185       - \int_eval:n { #2 + #3 }
3186     }
3187     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
3188     {
3189       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3190       \bool_set_true:N \l_@@_final_open_bool
3191     }
3192   }
3193   \group_begin:
3194   \int_compare:nNnTF { #1 } = 0
3195   { \color { nicematrix-first-row } }
3196   {
3197     \int_compare:nNnT { #1 } = \g_@@_row_total_int
3198     { \color { nicematrix-last-row } }
3199   }
3200   \keys_set:nn { NiceMatrix / xdots } { #4 }
3201   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3202   \@@_actually_draw_Ldots:
3203   \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).

```

3204   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
3205   { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
3206 }

3207 \AtBeginDocument
3208 {
3209   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3210   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3211   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
3212   {
3213     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
3214     {
3215       \@@_Vdotsfor:nnnn
3216       { \int_use:N \c@iRow }
3217       { \int_use:N \c@jCol }
3218       { #2 }
3219       {
3220         #1 , #3 ,
3221         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3222       }
3223     }
3224   }
3225 }

```

Enf of `\AtBeginDocument`.

```

3226 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
3227 {
3228   \bool_set_false:N \l_@@_initial_open_bool
3229   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

3230   \int_set:Nn \l_@@_initial_j_int { #2 }
3231   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```


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

```

3232 \int_compare:nNnTF #1 = 1
3233 {
3234   \int_set:Nn \l_@@_initial_i_int 1
3235   \bool_set_true:N \l_@@_initial_open_bool
3236 }
3237 {
3238   \cs_if_exist:cTF
3239   {
3240     pgf @ sh @ ns @ \@@_env:
3241     - \int_eval:n { #1 - 1 }
3242     - \int_use:N \l_@@_initial_j_int
3243   }
3244   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
3245   {
3246     \int_set:Nn \l_@@_initial_i_int { #1 }
3247     \bool_set_true:N \l_@@_initial_open_bool
3248   }
3249 }
3250 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
3251 {
3252   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3253   \bool_set_true:N \l_@@_final_open_bool
3254 }
3255 {
3256   \cs_if_exist:cTF
3257   {
3258     pgf @ sh @ ns @ \@@_env:
3259     - \int_eval:n { #1 + #3 }
3260     - \int_use:N \l_@@_final_j_int
3261   }
3262   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
3263   {
3264     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3265     \bool_set_true:N \l_@@_final_open_bool
3266   }
3267 }
3268 \group_begin:
3269 \int_compare:nNnTF { #2 } = 0
3270 { \color { nicematrix-first-col } }
3271 {
3272   \int_compare:nNnT { #2 } = \g_@@_col_total_int
3273   { \color { nicematrix-last-col } }
3274 }
3275 \keys_set:nn { NiceMatrix / xdots } { #4 }
3276 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3277 \@@_actually_draw_Vdots:
3278 \group_end:

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

3279 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
3280 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
3281 }

```

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

```

3282 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

```

The command `\line` accessible in `code-after`

In the `code-after`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells.

First, we write a command with an argument of the format i - j and applies the command `\int_eval:n` to i and j ; this must *not* be protected (and is, of course fully expandable).⁴⁷

```
3283 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
3284   { \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).

```
3285 \AtBeginDocument
3286 {
3287   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
3288   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3289   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
3290     {
3291       \group_begin:
3292       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
3293       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3294       \use:e
3295       {
3296         \@@_line_i:nn
3297         { \@@_double_int_eval:n #2 \q_stop }
3298         { \@@_double_int_eval:n #3 \q_stop }
3299       }
3300       \group_end:
3301     }
3302 }

3303 \cs_new_protected:Npn \@@_line_i:nn #1 #2
3304 {
3305   \bool_set_false:N \l_@@_initial_open_bool
3306   \bool_set_false:N \l_@@_final_open_bool
3307   \bool_if:nTF
3308   {
3309     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
3310     ||
3311     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
3312   }
3313   {
3314     \@@_error:nnn { unknown~cell~for~line~in~code~after } { #1 } { #2 }
3315   }
3316   { \@@_draw_line_ii:nn { #1 } { #2 } }
3317 }

3318 \AtBeginDocument
3319 {
3320   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
3321   {
```

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:.`

```
3322   \c_@@_pgfortikzpicture_tl
3323   \@@_draw_line_iii:nn { #1 } { #2 }
3324   \c_@@_endpgfortikzpicture_tl
3325 }
3326 }
```

⁴⁷Indeed, we want that the user may use the command `\line` in `code-after` with LaTeX counters in the arguments — with the command `\value`.

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

```

3327 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
3328 {
3329   \pgfrememberpicturepositiononpagetrue
3330   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
3331   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3332   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3333   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
3334   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3335   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3336   \@@_draw_line:
3337 }

```

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

Colors of cells, rows and columns

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

The commands `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor` (which are linked to `\rowcolor`, `\columncolor` and `\rectanglecolor` before the execution of the `code-before`) don't directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence `\l_@@_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 `\l_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\l_@@_color_i_tl`. In that token list, the instructions will `\@@_rowcolor:n`, `\@@_columncolor:n` and `\@@_rectanglecolor:nn` (corresponding of `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor`).

`bigskip #1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_color_seq` doesn't only add a color to `\l_@@_colors_seq`: it also updates the corresponding token list `\l_@@_color_i_tl`.

```

3338 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
3339 {

```

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.

```

3340   \int_zero:N \l_tmpa_int
3341   \seq_map_indexed_inline:Nn \l_@@_colors_seq
3342     { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
3343   \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

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

```

3344     {
3345       \seq_put_right:Nn \l_@@_colors_seq { #1 }
3346       \tl_set:cn { l_@@_color _ \seq_count:N \l_@@_colors_seq _ tl } { #2 }
3347     }

```

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

```

3348     { \tl_put_right:cn { l_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
3349   }

```

```

3350 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```

3351 \cs_new_protected:Npn \@@_actually_color:
3352 {
3353   \pgfpicture
3354   \pgf@relevantforpicturesizefalse
3355   \seq_map_indexed_inline:Nn \l_@@_colors_seq
3356   {
3357     \color ##2
3358     \use:c { l_@@_color _ ##1 _tl }
3359     \pgfusepathqfill
3360   }
3361   \endpgfpicture
3362 }
3363 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
3364 {
3365   \tl_set:Nn \l_tmpa_tl { #1 }
3366   \tl_set:Nn \l_tmpb_tl { #2 }
3367 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

3368 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
3369 {
3370   \tl_if_blank:nF { #2 }
3371   {
3372     \@@_add_to_colors_seq:xn
3373     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3374     { \@@_rowcolor:n { #3 } }
3375   }
3376 }
3377 \cs_new_protected:Npn \@@_rowcolor:n #1
3378 {
3379   \tl_set:Nn \l_@@_rows_tl { #1 }
3380   \tl_set:Nn \l_@@_cols_tl { - }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

3381   \@@_cartesian_path:
3382 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

3383 \NewDocumentCommand \@@_columncolor { 0 { } m m }
3384 {
3385   \tl_if_blank:nF { #2 }
3386   {
3387     \@@_add_to_colors_seq:xn
3388     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3389     { \@@_columncolor:n { #3 } }
3390   }
3391 }
3392 \cs_new_protected:Npn \@@_columncolor:n #1
3393 {
3394   \tl_set:Nn \l_@@_rows_tl { - }
3395   \tl_set:Nn \l_@@_cols_tl { #1 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

3396   \@@_cartesian_path:
3397 }

```

Here is an example : \@@_rectanglecolor{red!15}{2-3}{5-6}

```

3398 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
3399 {
3400   \tl_if_blank:nF { #2 }
3401   {
3402     \@@_add_to_colors_seq:xn
3403     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3404     { \@@_rectanglecolor:nn { #3 } { #4 } }
3405   }
3406 }

3407 \cs_new_protected:Npn \@@_rectanglecolor:nn #1 #2
3408 {
3409   \@@_cut_on_hyphen:w #1 \q_stop
3410   \tl_clear_new:N \l_tmpc_tl
3411   \tl_clear_new:N \l_tmpd_tl
3412   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
3413   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
3414   \@@_cut_on_hyphen:w #2 \q_stop
3415   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
3416   \tl_set:Nx \l_@@_cols_tl { \l_tmpd_tl - \l_tmpb_tl }

```

The command \@@_cartesian_path: takes in two implicit arguments: \l_@@_cols_tl and \l_@@_row_tl.

```

3417   \@@_cartesian_path:
3418 }

```

Here is an example : \@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}

```

3419 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
3420 {
3421   \clist_map_inline:nn { #3 }
3422   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
3423 }

```

```

3424 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
3425 {
3426   \int_step_inline:nn { \int_use:N \c@iRow }
3427   {
3428     \int_step_inline:nn { \int_use:N \c@jCol }
3429     {
3430       \int_if_even:nTF { #####1 + ##1 }
3431       { \@@_cellcolor [ #1 ] { #2 } }
3432       { \@@_cellcolor [ #1 ] { #3 } }
3433       { ##1 - #####1 }
3434     }
3435   }
3436 }

```

```

3437 \keys_define:nn { NiceMatrix / rowcolors }
3438 {
3439   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
3440   respect-blocks .default:n = true ,
3441   cols .tl_set:N = \l_@@_cols_tl ,
3442   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
3443   restart .default:n = true ,
3444   unknown .code:n = \@@_error:n { Unknown~option~for~rowcolors }
3445 }

```

The command `\rowcolors` (accessible in the `code-before`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`. Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

#1 (optional) is the color space ; #2 is a list of intervals of rows ; #3 is the first color ; #4 is the second color ; #5 is for the optional list of pairs key-value.

```
3446 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
3447 {
```

The group is for the options.

```
3448 \group_begin:
3449 \tl_clear_new:N \l_@@_cols_tl
3450 \tl_set:Nn \l_@@_cols_tl { - }
3451 \keys_set:nn { NiceMatrix / rowcolors } { #5 }
```

The boolean `\l_tmpa_bool` will indicate whereas we are in a row of the first color or of the second color.

```
3452 \bool_set_true:N \l_tmpa_bool
3453 \bool_lazy_and:nnT
3454 \l_@@_respect_blocks_bool
3455 {
3456 \cs_if_exist_p:c
3457 { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } }
3458 {
```

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`).

```
3459 \seq_set_eq:Nc \l_tmpb_seq
3460 { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
3461 \seq_set_filter:Nn \l_tmpa_seq \l_tmpb_seq
3462 { \@@_not_in_exterior_p:nnnn ##1 }
3463 }
3464 \pgfpicture
3465 \pgf@relevantforpicturesizefalse
3466 \clist_map_inline:nn { #2 }
3467 {
3468 \tl_set:Nn \l_tmpa_tl { ##1 }
3469 \tl_if_in:NnTF \l_tmpa_tl { - }
3470 { \@@_cut_on_hyphen:w ##1 \q_stop }
3471 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
```

The counter `\l_tmpa_int` will be the index of the loop.

```
3472 \int_set:Nn \l_tmpa_int \l_tmpa_tl
3473 \bool_if:NTF \l_@@_rowcolors_restart_bool
3474 { \bool_set_true:N \l_tmpa_bool }
3475 { \bool_set:Nn \l_tmpa_bool { \int_if_odd_p:n { \l_tmpa_tl } } }
3476 \int_zero_new:N \l_tmpc_int
3477 \int_set:Nn \l_tmpc_int \l_tmpb_tl
3478 \int_do_until:nNnn \l_tmpa_int > \l_tmpc_int
3479 {
```

We will compute in `\l_tmpb_int` the last row of the “block”.

```
3480 \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).

```
3481 \bool_lazy_and:nnT
3482 \l_@@_respect_blocks_bool
3483 {
3484 \cs_if_exist_p:c
3485 { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
3486 }
3487 {
3488 \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
3489 { \@@_intersect_our_row_p:nnnn ####1 }
3490 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn ####1 }
```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

3491     }
3492     \tl_set:Nx \l_@@_rows_tl
3493     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
3494     \bool_if:NTF \l_tmpa_bool
3495     {
3496         \tl_if_blank:nF { #3 }
3497         {
3498             \tl_if_empty:nTF { #1 }
3499             \color
3500             { \color [ #1 ] }
3501             { #3 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

3502         \@@_cartesian_path:
3503         \pgfusepathqfill
3504     }
3505     \bool_set_false:N \l_tmpa_bool
3506 }
3507 {
3508     \tl_if_blank:nF { #4 }
3509     {
3510         \tl_if_empty:nTF { #1 }
3511         \color
3512         { \color [ #1 ] }
3513         { #4 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

3514         \@@_cartesian_path:
3515         \pgfusepathqfill
3516     }
3517     \bool_set_true:N \l_tmpa_bool
3518 }
3519     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
3520 }
3521 }
3522 \endpgfpicture
3523 \group_end:
3524 }

```

```

3525 \cs_new_protected:Npn \@@_rowcolors_i:nnnn #1 #2 #3 #4
3526 {
3527     \int_compare:nNnT { #3 } > \l_tmpb_int
3528     { \int_set:Nn \l_tmpb_int { #3 } }
3529 }

```

```

3530 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnn p
3531 {
3532     \bool_lazy_or:nnTF
3533     { \int_compare_p:nNn { #4 } = \c_zero_int }
3534     { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
3535     \prg_return_false:
3536     \prg_return_true:
3537 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

3538 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnn p
3539 {
3540     \bool_if:nTF

```

```

3541 {
3542   \int_compare_p:n { #1 <= \l_tmpa_int }
3543   &&
3544   \int_compare_p:n { \l_tmpa_int <= #3 }
3545 }
3546 \prg_return_true:
3547 \prg_return_false:
3548 }

```

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.

```

3549 \cs_new_protected:Npn \@@_cartesian_path:
3550 {

```

We begin the loop over the columns.

```

3551   \clist_map_inline:Nn \l_@@_cols_tl
3552   {
3553     \tl_set:Nn \l_tmpa_tl { ##1 }
3554     \tl_if_in:NnTF \l_tmpa_tl { - }
3555     { \@@_cut_on_hyphen:w ##1 \q_stop }
3556     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
3557     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
3558     \tl_if_empty:NT \l_tmpb_tl
3559     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
3560     \int_compare:nNnT \l_tmpb_tl > \c@jCol
3561     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
3562     \@@_qpoint:n { col - \l_tmpa_tl }
3563     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
3564     { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
3565     { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
3566     \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
3567     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows

```

3568   \clist_map_inline:Nn \l_@@_rows_tl
3569   {
3570     \tl_set:Nn \l_tmpa_tl { ####1 }
3571     \tl_if_in:NnTF \l_tmpa_tl { - }
3572     { \@@_cut_on_hyphen:w ####1 \q_stop }
3573     { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
3574     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
3575     \tl_if_empty:NT \l_tmpb_tl
3576     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
3577     \int_compare:nNnT \l_tmpb_tl > \c@iRow
3578     { \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`.

```

3579     \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
3580     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
3581     \@@_qpoint:n { row - \l_tmpa_tl }
3582     \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
3583     \pgfpathrectanglecorners
3584     { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
3585     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3586   }
3587 }
3588 }

```

When the user uses the key `colortbl-like`, the following command will be linked to `\cellcolor` in the tabular.

```

3589 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
3590 {

```



```

3591 \tl_gput_right:Nx \g_nicematrix_code_before_tl
3592 { \cellcolor [ #1 ] { #2 } { \int_use:N \c@iRow - \int_use:N \c@jCol } }
3593 }

```

When the user uses the key `colortbll`-like, the following command will be linked to `\rowcolor` in the tabular.

```

3594 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
3595 {
3596   \tl_gput_right:Nx \g_nicematrix_code_before_tl
3597   {
3598     \exp_not:N \rectanglecolor [ #1 ] { #2 }
3599     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3600     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
3601   }
3602 }

```

```

3603 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
3604 {
3605   \int_compare:nNnT \c@iRow = 1
3606   {

```

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).

```

3607   \tl_gput_left:Nx \g_nicematrix_code_before_tl
3608   { \exp_not:N \columncolor [ #1 ] { #2 } { \int_use:N \c@jCol } }
3609 }
3610 }

```

The vertical rules

We give to the user the possibility to define new types of columns (with `\newcolumnstype` 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`.

```

3611 \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`.

```

3612 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
3613 {
3614   \int_compare:nNnTF \l_@@_first_col_int = 0
3615   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
3616   {
3617     \int_compare:nNnTF \c@jCol = 0
3618     {
3619       \int_compare:nNnF \c@iRow = { -1 }
3620       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
3621     }
3622     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
3623   }
3624 }

```

This definition may seem complicated by we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* an 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.

```

3625 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
3626 {
3627   \int_compare:nNnF \c@iRow = 0
3628     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
3629 }
```

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

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the column `#1`. `#2` is the number of consecutive occurrences of `|`.

```

3630 \cs_new_protected:Npn \@@_vline:nn #1 #2
3631 {
```

The following test is for the case where the user don't use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

3632   \int_compare:nNnT { #1 } < { \c@jCol + 2 }
3633   {
3634     \pgfpicture
3635       \@@_vline_i:nn { #1 } { #2 }
3636     \endpgfpicture
3637   }
3638 }
```

```

3639 \cs_new_protected:Npn \@@_vline_i:nn #1 #2
3640 {
```

`\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_tmptc_tl`.

```

3641   \tl_set:Nx \l_tmpb_tl { #1 }
3642   \tl_clear_new:N \l_tmptc_tl
3643   \int_step_variable:nNn \c@iRow \l_tmpa_tl
3644   {
```

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.

```

3645     \bool_gset_true:N \g_tmpa_bool
3646     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3647       { \@@_test_if_vline_in_block:nnnn #1 }
3648     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
3649       { \@@_test_if_vline_in_block:nnnn #1 }
3650     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
3651       { \@@_test_if_vline_in_stroken_block:nnnn #1 }
3652     \clist_if_empty:NF \l_@@_except_corners_clist
3653     \@@_test_in_corner_v:
3654     \bool_if:NTF \g_tmpa_bool
3655     {
3656       \tl_if_empty:NT \l_tmptc_tl
```

We keep in memory that we have a rule to draw.

```

3657       { \tl_set_eq:NN \l_tmptc_tl \l_tmpa_tl }
3658     }
3659   {
3660     \tl_if_empty:NF \l_tmptc_tl
3661     {
3662       \@@_vline_ii:nnnn
3663       { #1 }
```

```

3664         { #2 }
3665         \l_tmpc_tl
3666         { \int_eval:n { \l_tmpa_tl - 1 } }
3667         \tl_clear:N \l_tmpc_tl
3668     }
3669 }
3670 }
3671 \tl_if_empty:NF \l_tmpc_tl
3672 {
3673     \@@_vline_ii:nnnn
3674     { #1 }
3675     { #2 }
3676     \l_tmpc_tl
3677     { \int_use:N \c@iRow }
3678     \tl_clear:N \l_tmpc_tl
3679 }
3680 }

3681 \cs_new_protected:Npn \@@_test_in_corner_v:
3682 {
3683     \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
3684     {
3685         \seq_if_in:NxT
3686         \l_@@_empty_corner_cells_seq
3687         { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
3688         { \bool_set_false:N \g_tmpa_bool }
3689     }
3690     {
3691         \seq_if_in:NxT
3692         \l_@@_empty_corner_cells_seq
3693         { \l_tmpa_tl - \l_tmpb_tl }
3694         {
3695             \int_compare:nNnTF \l_tmpb_tl = 1
3696             { \bool_set_false:N \g_tmpa_bool }
3697             {
3698                 \seq_if_in:NxT
3699                 \l_@@_empty_corner_cells_seq
3700                 { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
3701                 { \bool_set_false:N \g_tmpa_bool }
3702             }
3703         }
3704     }
3705 }

```

#1 is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the number of the rows between which the rule has to be drawn.

```

3706 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
3707 {
3708     \pgfrememberpicturepositiononpagetrue
3709     \pgf@relevantforpicturesizefalse
3710     \@@_qpoint:n { row - #3 }
3711     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3712     \@@_qpoint:n { col - #1 }
3713     \dim_set_eq:NN \l_tmpb_dim \pgf@x
3714     \@@_qpoint:n { row - \@@_succ:n { #4 } }
3715     \dim_set_eq:NN \l_tmpc_dim \pgf@y
3716     \bool_lazy_and:nnT
3717     { \int_compare_p:nNn { #2 } > 1 }
3718     { ! \tl_if_blank_p:V \CT@drsc@ }
3719     {
3720         \group_begin:
3721         \CT@drsc@

```

```

3722 \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
3723 \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
3724 \dim_set:Nn \l_tmpd_dim
3725 { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
3726 \pgfpathrectanglecorners
3727 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
3728 { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
3729 \pgfusepathqfill
3730 \group_end:
3731 }
3732 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
3733 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
3734 \prg_replicate:nn { #2 - 1 }
3735 {
3736 \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
3737 \dim_sub:Nn \l_tmpb_dim \doublerulesep
3738 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
3739 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
3740 }
3741 \CT@arc@
3742 \pgfsetlinewidth { 1.1 \arrayrulewidth }
3743 \pgfsetrectcap
3744 \pgfusepathqstroke
3745 }

```

The following draws a complete vertical rule in the column #1 (#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key `except-corners` is not used).

```

3746 \cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
3747 { \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

```

The command `\@@_draw_hlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `except-corners` is used).

```

3748 \cs_new_protected:Npn \@@_draw_vlines:
3749 {
3750 \int_step_inline:nnn
3751 { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
3752 { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
3753 { \@@_vline:nn { ##1 } 1 }
3754 }

```

The horizontal rules

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the row #1. #2 is the number of consecutive occurrences of `\Hline`.

```

3755 \cs_new_protected:Npn \@@_hline:nn #1 #2
3756 {
3757 \pgfpicture
3758 \@@_hline_i:nn { #1 } { #2 }
3759 \endpgfpicture
3760 }
3761 \cs_new_protected:Npn \@@_hline_i:nn #1 #2
3762 {

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. Whe, we have found a column corresponding to a rule to draw, we note its number in `\l_tmpc_tl`.

```

3763 \tl_set:Nn \l_tmpa_tl { #1 }
3764 \tl_clear_new:N \l_tmpc_tl
3765 \int_step_variable:nNn \c@jCol \l_tmpb_tl
3766 {

```

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.

```

3767     \bool_gset_true:N \g_tmpa_bool
3768     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3769       { \@@_test_if_hline_in_block:nnnn ##1 }
3770     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
3771       { \@@_test_if_hline_in_block:nnnn ##1 }
3772     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
3773       { \@@_test_if_hline_in_stroken_block:nnnn ##1 }
3774     \clist_if_empty:NF \l_@@_except_corners_clist \@@_test_in_corner_h:
3775     \bool_if:NTF \g_tmpa_bool
3776       {
3777         \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

3778         { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
3779       }
3780     {
3781       \tl_if_empty:NF \l_tmpc_tl
3782       {
3783         \@@_hline_ii:nnnn
3784         { #1 }
3785         { #2 }
3786         \l_tmpc_tl
3787         { \int_eval:n { \l_tmpb_tl - 1 } }
3788         \tl_clear:N \l_tmpc_tl
3789       }
3790     }
3791   }
3792   \tl_if_empty:NF \l_tmpc_tl
3793   {
3794     \@@_hline_ii:nnnn
3795     { #1 }
3796     { #2 }
3797     \l_tmpc_tl
3798     { \int_use:N \c@jCol }
3799     \tl_clear:N \l_tmpc_tl
3800   }
3801 }

```

```

3802 \cs_new_protected:Npn \@@_test_in_corner_h:
3803 {
3804   \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
3805   {
3806     \seq_if_in:NxT
3807       \l_@@_empty_corner_cells_seq
3808       { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
3809       { \bool_set_false:N \g_tmpa_bool }
3810   }
3811   {
3812     \seq_if_in:NxT
3813       \l_@@_empty_corner_cells_seq
3814       { \l_tmpa_tl - \l_tmpb_tl }
3815     {
3816       \int_compare:nNnTF \l_tmpa_tl = 1
3817       { \bool_set_false:N \g_tmpa_bool }
3818       {
3819         \seq_if_in:NxT
3820           \l_@@_empty_corner_cells_seq
3821           { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
3822           { \bool_set_false:N \g_tmpa_bool }

```

```

3823     }
3824   }
3825 }
3826 }

```

#1 is the number of the row; #2 is the number of horizontal rules to draw (with potentially a color between); #3 and #4 are the number of the columns between which the rule has to be drawn.

```

3827 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
3828 {
3829   \pgfrememberpicturepositiononpagetrue
3830   \pgf@relevantforpicturesizefalse
3831   \@@_qpoint:n { col - #3 }
3832   \dim_set_eq:NN \l_tmpa_dim \pgf@x
3833   \@@_qpoint:n { row - #1 }
3834   \dim_set_eq:NN \l_tmpb_dim \pgf@y
3835   \@@_qpoint:n { col - \@@_succ:n { #4 } }
3836   \dim_set_eq:NN \l_tmpc_dim \pgf@x
3837   \bool_lazy_and:nnT
3838     { \int_compare_p:nNn { #2 } > 1 }
3839     { ! \tl_if_blank_p:V \CT@drsc@ }
3840     {
3841       \group_begin:
3842       \CT@drsc@
3843       \dim_set:Nn \l_tmpd_dim
3844         { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
3845       \pgfpathrectanglecorners
3846         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3847         { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
3848       \pgfusepathqfill
3849       \group_end:
3850     }
3851   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3852   \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
3853   \prg_replicate:nn { #2 - 1 }
3854   {
3855     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
3856     \dim_sub:Nn \l_tmpb_dim \doublerulesep
3857     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3858     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
3859   }
3860   \CT@arc@
3861   \pgfsetlinewidth { 1.1 \arrayrulewidth }
3862   \pgfsetrectcap
3863   \pgfusepathqstroke
3864 }

```

```

3865 \cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
3866 { \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as `\Cdots` and in the corners (if the key `except-corners` is used).

```

3867 \cs_new_protected:Npn \@@_draw_hlines:
3868 {
3869   \int_step_inline:nnn
3870     { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
3871     { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
3872     { \@@_hline:nn { ##1 } 1 }
3873 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

3874 \cs_set:Npn \@@_Hline: { \noalign { \ifnum 0 = ` } \fi \@@_Hline_i:n { 1 } }

```

The argument of the command `\@@_Hline_i:n` is the number of successive `\Hline` found.

```

3875 \cs_set:Npn \@@_Hline_i:n #1
3876 {
3877   \peek_meaning_ignore_spaces:NTF \Hline
3878   { \@@_Hline_ii:nn { #1 + 1 } }
3879   { \@@_Hline_iii:n { #1 } }
3880 }
3881 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
3882 \cs_set:Npn \@@_Hline_iii:n #1
3883 {
3884   \skip_vertical:n
3885   {
3886     \arrayrulewidth * ( #1 )
3887     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
3888   }
3889   \tl_gput_right:Nx \g_@@_internal_code_after_tl
3890   { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
3891   \ifnum 0 = `{ \fi }
3892 }

```

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 col) 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.

```

3893 \cs_new_protected:Npn \@@_test_if_hline_in_block:nnnn #1 #2 #3 #4
3894 {
3895   \bool_lazy_all:nT
3896   {
3897     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
3898     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
3899     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
3900     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
3901   }
3902   { \bool_gset_false:N \g_tmpa_bool }
3903 }

```

The same for vertical rules.

```

3904 \cs_new_protected:Npn \@@_test_if_vline_in_block:nnnn #1 #2 #3 #4
3905 {
3906   \bool_lazy_all:nT
3907   {
3908     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
3909     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
3910     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
3911     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
3912   }
3913   { \bool_gset_false:N \g_tmpa_bool }
3914 }
3915 \cs_new_protected:Npn \@@_test_if_hline_in_stroken_block:nnnn #1 #2 #3 #4
3916 {
3917   \bool_lazy_all:nT
3918   {
3919     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
3920     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
3921     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
3922     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
3923   }
3924   { \bool_gset_false:N \g_tmpa_bool }
3925 }

```

```

3926 \cs_new_protected:Npn \@@_test_if_vline_in_stroken_block:nnnn #1 #2 #3 #4
3927 {
3928   \bool_lazy_all:nT
3929   {
3930     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
3931     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
3932     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
3933     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
3934   }
3935   { \bool_gset_false:N \g_tmpa_bool }
3936 }

```

The key except-corners

When the key `except-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.

```

3937 \cs_new_protected:Npn \@@_compute_corners:
3938 {

```

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

```

3939   \seq_clear_new:N \l_@@_empty_corner_cells_seq
3940   \clist_map_inline:Nn \l_@@_except_corners_clist
3941   {
3942     \str_case:nnF { ##1 }
3943     {
3944       { NW }
3945       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
3946       { NE }
3947       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
3948       { SW }
3949       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
3950       { SE }
3951       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
3952     }
3953     { \@@_error:nn { bad-corner } { ##1 } }
3954   }
3955 }

```

“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_@@_empty_corner_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.

```

3956 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
3957 {

```

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.

```

3958   \bool_set_false:N \l_tmpa_bool
3959   \int_zero_new:N \l_@@_last_empty_row_int
3960   \int_set:Nn \l_@@_last_empty_row_int { #1 }

```



```

3961 \int_step_inline:nnnn { #1 } { #3 } { #5 }
3962 {
3963   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
3964   \bool_lazy_or:nnTF
3965   {
3966     \cs_if_exist_p:c
3967     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
3968   }
3969   \l_tmpb_bool
3970   { \bool_set_true:N \l_tmpa_bool }
3971   {
3972     \bool_if:NF \l_tmpa_bool
3973     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
3974   }
3975 }

```

Now, you determine the last empty cell in the row of number 1.

```

3976 \bool_set_false:N \l_tmpa_bool
3977 \int_zero_new:N \l_@@_last_empty_column_int
3978 \int_set:Nn \l_@@_last_empty_column_int { #2 }
3979 \int_step_inline:nnnn { #2 } { #4 } { #6 }
3980 {
3981   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
3982   \bool_lazy_or:nnTF
3983   \l_tmpb_bool
3984   {
3985     \cs_if_exist_p:c
3986     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
3987   }
3988   { \bool_set_true:N \l_tmpa_bool }
3989   {
3990     \bool_if:NF \l_tmpa_bool
3991     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
3992   }
3993 }

```

Now, we loop over the rows.

```

3994 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
3995 {

```

We treat the row number ##1 with another loop.

```

3996   \bool_set_false:N \l_tmpa_bool
3997   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
3998   {
3999     \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
4000     \bool_lazy_or:nnTF
4001     \l_tmpb_bool
4002     {
4003       \cs_if_exist_p:c
4004       { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
4005     }
4006     { \bool_set_true:N \l_tmpa_bool }
4007     {
4008       \bool_if:NF \l_tmpa_bool
4009       {
4010         \int_set:Nn \l_@@_last_empty_column_int { #####1 }
4011         \seq_put_right:Nn
4012         \l_@@_empty_corner_cells_seq
4013         { ##1 - #####1 }
4014       }
4015     }
4016   }
4017 }
4018 }

```

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`).

```

4019 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
4020 {
4021   \int_set:Nn \l_tmpa_int { #1 }
4022   \int_set:Nn \l_tmpb_int { #2 }
4023   \bool_set_false:N \l_tmpb_bool
4024   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4025     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
4026 }
4027 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6
4028 {
4029   \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
4030   {
4031     \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
4032     {
4033       \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
4034       {
4035         \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
4036         { \bool_set_true:N \l_tmpb_bool }
4037       }
4038     }
4039   }
4040 }

```

The commands to draw dotted lines to separate columns and rows

These commands don't use the normal nodes, the medium nor the large nodes. They only use the `col` nodes and the `row` nodes.

Horizontal dotted lines

The following command must *not* be protected because it's meant to be expanded in a `\noalign`.

```

4041 \cs_new:Npn \@@_hdottedline:
4042 {
4043   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
4044   \@@_hdottedline_i:
4045 }

```

On the other side, the following command should be protected.

```

4046 \cs_new_protected:Npn \@@_hdottedline_i:
4047 {

```

We write in the code-after the instruction that will actually draw the dotted line. It's not possible to draw this dotted line now because we don't know the length of the line (we don't even know the number of columns).

```

4048   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4049   { \@@_hdottedline:n { \int_use:N \c@iRow } }
4050 }

```

The command `\@@_hdottedline:n` is the command written in the code-after that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```

4051 \AtBeginDocument
4052 {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That's why we construct now a version of `\@@_hdottedline:n` with the right environment (`\begin{pgfpicture}\end{pgfpicture}` or `\begin{tikzpicture}...\end{tikzpicture}`).

```

4053   \cs_new_protected:Npx \@@_hdottedline:n #1

```

```

4054 {
4055   \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
4056   \bool_set_true:N \exp_not:N \l_@@_final_open_bool
4057   \c_@@_pgfortikzpicture_tl
4058   \@@_hdottedline_i:n { #1 }
4059   \c_@@_endpgfortikzpicture_tl
4060 }
4061 }

```

The following command *must* be protected since it is used in the construction of `\@@_hdottedline:n`.

```

4062 \cs_new_protected:Npn \@@_hdottedline_i:n #1
4063 {
4064   \pgfrememberpicturepositiononpagetrue
4065   \@@_qpoint:n { row - #1 }

```

We do a translation par `-\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```

4066   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4067   \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
4068   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim

```

The dotted line will be extended if the user uses `margin` (or `left-margin` and `right-margin`).

The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn’t).

```
\begin{bNiceMatrix}
```

```
1 & 2 & 3 & 4 \\\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
```

```
1 & 2 & 3 & 4 \\\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

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

```

4069   \@@_qpoint:n { col - 1 }
4070   \dim_set:Nn \l_@@_x_initial_dim
4071   {
4072     \pgf@x +

```

We do a reduction by `\arraycolsep` for the environments with delimiters (and not for the other).

```

4073   \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4074   - \l_@@_left_margin_dim
4075   }
4076   \@@_qpoint:n { col - \@@_succ:n \c_jCol }
4077   \dim_set:Nn \l_@@_x_final_dim
4078   {
4079     \pgf@x -
4080     \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4081     + \l_@@_right_margin_dim
4082   }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by `0.5 \l_@@_inter_dots_dim` is *ad hoc* for a better result.

```

4083   \tl_set:Nn \l_tmpa_tl { ( }
4084   \tl_if_eq:NnF \l_@@_left_delim_tl \l_tmpa_tl
4085   { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
4086   \tl_set:Nn \l_tmpa_tl { ) }

```

```

4087 \tl_if_eq:NNF \l_@@_right_delim_tl \l_tmpa_tl
4088 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }

```

As of now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

4089 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4090 \@@_draw_line:
4091 }

```

Vertical dotted lines

```

4092 \cs_new_protected:Npn \@@_vdottedline:n #1
4093 {
4094   \bool_set_true:N \l_@@_initial_open_bool
4095   \bool_set_true:N \l_@@_final_open_bool

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”.

```

4096   \bool_if:NTF \c_@@_tikz_loaded_bool
4097   {
4098     \tikzpicture
4099     \@@_vdottedline_i:n { #1 }
4100     \endtikzpicture
4101   }
4102   {
4103     \pgfpicture
4104     \@@_vdottedline_i:n { #1 }
4105     \endpgfpicture
4106   }
4107 }

```

```

4108 \cs_new_protected:Npn \@@_vdottedline_i:n #1
4109 {

```

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.

```

4110   \CT@arc@
4111   \pgfrememberpicturepositiononpagetrue
4112   \@@_qpoint:n { col - \int_eval:n { #1 + 1 } }

```

We do a translation par `-\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```

4113   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
4114   \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
4115   \@@_qpoint:n { row - 1 }

```

We arbitrary decrease the height of the dotted line by a quantity equal to `\l_@@_inter_dots_dim` in order to improve the visual impact.

```

4116   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
4117   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
4118   \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }

```

As of now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

4119   \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4120   \@@_draw_line:
4121 }

```

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.

```
4122 \bool_new:N \l_@@_block_auto_columns_width_bool
```

As of now, there is only one option available for the environment {NiceMatrixBlock}.

```
4123 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
4124 {
4125   auto-columns-width .code:n =
4126   {
4127     \bool_set_true:N \l_@@_block_auto_columns_width_bool
4128     \dim_gzero_new:N \g_@@_max_cell_width_dim
4129     \bool_set_true:N \l_@@_auto_columns_width_bool
4130   }
4131 }

4132 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
4133 {
4134   \int_gincr:N \g_@@_NiceMatrixBlock_int
4135   \dim_zero:N \l_@@_columns_width_dim
4136   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
4137   \bool_if:NT \l_@@_block_auto_columns_width_bool
4138   {
4139     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4140     {
4141       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
4142       { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
4143     }
4144   }
4145 }
```

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).

```
4146 {
4147   \bool_if:NT \l_@@_block_auto_columns_width_bool
4148   {
4149     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
4150     \iow_shipout:Nx \@mainaux
4151     {
4152       \cs_gset:cpn
4153       { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```
4154       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
4155     }
4156     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
4157   }
4158 }
```

The extra nodes

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

```
4159 \cs_generate_variant:Nn \dim_min:nn { v n }
4160 \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 dans that construction uses the standard medium nodes).

```

4161 \cs_new_protected:Npn \@@_create_extra_nodes:
4162 {
4163   \bool_if:NTF \l_@@_medium_nodes_bool
4164   {
4165     \bool_if:NTF \l_@@_large_nodes_bool
4166     \@@_create_medium_and_large_nodes:
4167     \@@_create_medium_nodes:
4168   }
4169   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
4170 }

```

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`.

```

4171 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
4172 {
4173   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4174   {
4175     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
4176     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
4177     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
4178     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
4179   }
4180   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4181   {
4182     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
4183     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
4184     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
4185     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
4186   }

```

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

```

4187   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4188   {
4189     \int_step_variable:nnNn
4190     \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.

```

4191     {
4192       \cs_if_exist:cT
4193       { 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`.

```

4194         {
4195             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
4196             \dim_set:cn { l_@@_row_\@@_i: _min_dim}
4197             { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
4198             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
4199             {
4200                 \dim_set:cn { l_@@_column _ \@@_j: _min_dim}
4201                 { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
4202             }

```

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`.

```

4203             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
4204             \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
4205             { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
4206             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
4207             {
4208                 \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
4209                 { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
4210             }
4211         }
4212     }
4213 }

```

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

```

4214 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4215 {
4216     \dim_compare:nNnT
4217     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
4218     {
4219         \@@_qpoint:n { row - \@@_i: - base }
4220         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
4221         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
4222     }
4223 }
4224 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4225 {
4226     \dim_compare:nNnT
4227     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
4228     {
4229         \@@_qpoint:n { col - \@@_j: }
4230         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
4231         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
4232     }
4233 }
4234 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

4235 \cs_new_protected:Npn \@@_create_medium_nodes:
4236 {
4237     \pgfpicture
4238     \pgfrememberpicturepositiononpagetrue
4239     \pgf@relevantforpicturesizefalse
4240     \@@_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”.

```

4241     \tl_set:Nn \l_@@_suffix_tl { -medium }
4242     \@@_create_nodes:

```

```

4243 \endpgfpicture
4244 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁴⁸. 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:`.

```

4245 \cs_new_protected:Npn \@@_create_large_nodes:
4246 {
4247   \pgfpicture
4248   \pgfrememberpicturepositiononpagetrue
4249   \pgf@relevantforpicturesizefalse
4250   \@@_computations_for_medium_nodes:
4251   \@@_computations_for_large_nodes:
4252   \tl_set:Nn \l_@@_suffix_tl { - large }
4253   \@@_create_nodes:
4254   \endpgfpicture
4255 }

4256 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
4257 {
4258   \pgfpicture
4259   \pgfrememberpicturepositiononpagetrue
4260   \pgf@relevantforpicturesizefalse
4261   \@@_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”.

```

4262   \tl_set:Nn \l_@@_suffix_tl { - medium }
4263   \@@_create_nodes:
4264   \@@_computations_for_large_nodes:
4265   \tl_set:Nn \l_@@_suffix_tl { - large }
4266   \@@_create_nodes:
4267   \endpgfpicture
4268 }

```

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

4269 \cs_new_protected:Npn \@@_computations_for_large_nodes:
4270 {
4271   \int_set:Nn \l_@@_first_row_int 1
4272   \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`.

```

4273   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
4274   {
4275     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
4276     {
4277       (
4278         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
4279         \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4280       )
4281       / 2
4282     }
4283     \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4284     { l_@@_row_\@@_i: _ min_dim }
4285   }
4286   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
4287   {
4288     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }

```

⁴⁸If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`


```

4289     {
4290     (
4291         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
4292         \dim_use:c
4293         { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4294     )
4295     / 2
4296     }
4297     \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4298     { l_@@_column _ \@@_j: _ max _ dim }
4299 }

```

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

```

4300     \dim_sub:cn
4301     { l_@@_column _ 1 _ min _ dim }
4302     \l_@@_left_margin_dim
4303     \dim_add:cn
4304     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
4305     \l_@@_right_margin_dim
4306 }

```

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).

```

4307 \cs_new_protected:Npn \@@_create_nodes:
4308 {
4309     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4310     {
4311         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4312         {

```

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

```

4313         \@@_pgf_rect_node:nnnnn
4314         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4315         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
4316         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
4317         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
4318         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
4319         \str_if_empty:NF \l_@@_name_str
4320         {
4321             \pgfnodealias
4322             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4323             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4324         }
4325     }
4326 }

```

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 .

```

4327     \seq_mapthread_function:NNN
4328     \g_@@_multicolumn_cells_seq
4329     \g_@@_multicolumn_sizes_seq
4330     \@@_node_for_multicolumn:nn
4331 }

```

```

4332 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
4333 {
4334     \cs_set_nopar:Npn \@@_i: { #1 }
4335     \cs_set_nopar:Npn \@@_j: { #2 }
4336 }

```

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”).

```

4337 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
4338 {
4339   \@@_extract_coords_values: #1 \q_stop
4340   \@@_pgf_rect_node:nnnnn
4341     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4342     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
4343     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
4344     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
4345     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
4346   \str_if_empty:NF \l_@@_name_str
4347   {
4348     \pgfnodealias
4349       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4350       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
4351   }
4352 }

```

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.

```

4353 \keys_define:nn { NiceMatrix / Block / FirstPass }
4354 {
4355   l .code:n = \tl_set:Nn \l_@@_pos_of_block_tl l ,
4356   l .value_forbidden:n = true ,
4357   r .code:n = \tl_set:Nn \l_@@_pos_of_block_tl r ,
4358   r .value_forbidden:n = true ,
4359   c .code:n = \tl_set:Nn \l_@@_pos_of_block_tl c ,
4360   c .value_forbidden:n = true ,

```

The two following lines will be uncommented when we will give to the key `color` its new definition.

```

4361   % color .tl_set:N = \l_@@_color_tl ,
4362   % color .value_required:n = true ,
4363 }

```

The following command `\@@_Block:` will be linked to `\Block` in the environments of `nicematrix`. We define it with `\NewExpandableDocumentCommand` because it has an optional argument between `<` and `>` (for TeX instructions put before the math mode of the label and before the beginning of the small array of the block). It's mandatory to use an expandable command.

```

4364 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
4365 {

```

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).

```

4366   \tl_if_blank:nTF { #2 } { \@@_Block_i 1-1 \q_stop } { \@@_Block_i #2 \q_stop }
4367   { #1 } { #3 } { #4 }
4368 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

4369 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

Now, the arguments have been extracted: #1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of key-values, #4 are the tokens to put before the math mode and the beginning of the small array of the block and #5 is the label of the block.

```
4370 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
4371 {
```

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*).

```
4372   \bool_lazy_or:nnTF
4373     { \tl_if_blank_p:n { #1 } }
4374     { \str_if_eq_p:nn { #1 } { * } }
4375     { \int_set:Nn \l_tmpa_int { 100 } }
4376     { \int_set:Nn \l_tmpa_int { #1 } }
4377   \bool_lazy_or:nnTF
4378     { \tl_if_blank_p:n { #2 } }
4379     { \str_if_eq_p:nn { #2 } { * } }
4380     { \int_set:Nn \l_tmpb_int { 100 } }
4381     { \int_set:Nn \l_tmpb_int { #2 } }
4382   \int_compare:nNnTF \l_tmpb_int = 1
4383     {
4384       \tl_if_empty:NTF \l_@@_cell_type_tl
4385         { \tl_set:Nn \l_@@_pos_of_block_tl c }
4386         { \tl_set_eq:NN \l_@@_pos_of_block_tl \l_@@_cell_type_tl }
4387     }
4388     { \tl_set:Nn \l_@@_pos_of_block_tl c }
4389   \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
4390   \tl_set:Nx \l_tmpa_tl
4391     {
4392       { \int_use:N \c@iRow }
4393       { \int_use:N \c@jCol }
4394       { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
4395       { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
4396     }
```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets: `{imin}{jmin}{imax}{jmax}`.

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by currying).

```
4397   \bool_lazy_or:nnTF
4398     { \int_compare_p:nNn { \l_tmpa_int } = 1 }
4399     { \int_compare_p:nNn { \l_tmpb_int } = 1 }
4400     { \exp_args:Nxx \@@_Block_iv:nnnnn }
4401     { \exp_args:Nxx \@@_Block_v:nnnnn }
4402   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
4403 }
```

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).

```
4404 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
4405 {
4406   \int_gincr:N \g_@@_block_box_int
4407   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
```

```

4408 {
4409   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4410   {
4411     \@@_actually_diagbox:nnnnnn
4412     { \int_use:N \c@iRow }
4413     { \int_use:N \c@jCol }
4414     { \int_eval:n { \c@iRow + #1 - 1 } }
4415     { \int_eval:n { \c@jCol + #2 - 1 } }
4416     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
4417   }
4418 }
4419 \box_gclear_new:c
4420 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4421 \hbox_gset:cn
4422 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4423 {

```

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with `\set@color` and not `\color_ensure_current:` because that command seems to be bugged: it doesn't work in XeLaTeX when `fontspec` is loaded.

```

4424   \tl_if_empty:NTF \l_@@_color_tl
4425   { \int_compare:nNnT { #2 } = 1 \set@color }
4426   { \color { \l_@@_color_tl } }
4427   \group_begin:
4428   \cs_set:Npn \arraystretch { 1 }
4429   \dim_set_eq:NN \extrarowheight \c_zero_dim
4430   #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).

```

4431   \bool_if:NT \g_@@_rotate_bool { \tl_set:Nn \l_@@_pos_of_block_tl c }
4432   \bool_if:NTF \l_@@_NiceTabular_bool
4433   {
4434     \exp_args:Nnx \begin { tabular }
4435     { @ { } \l_@@_pos_of_block_tl @ { } }
4436     #5
4437     \end { tabular }
4438   }
4439   {
4440     \c_math_toggle_token
4441     \exp_args:Nnx \begin { array }
4442     { @ { } \l_@@_pos_of_block_tl @ { } }
4443     #5
4444     \end { array }
4445     \c_math_toggle_token
4446   }
4447   \group_end:
4448 }
4449 \bool_if:NT \g_@@_rotate_bool
4450 {
4451   \box_grotate:cn
4452   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4453   { 90 }
4454   \bool_gset_false:N \g_@@_rotate_bool
4455 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

4456   \int_compare:nNnT { #2 } = 1
4457   {

```

```

4458 \dim_gset:Nn \g_@@_blocks_wd_dim
4459 {
4460   \dim_max:nn
4461   \g_@@_blocks_wd_dim
4462   {
4463     \box_wd:c
4464     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4465   }
4466 }
4467 }

```

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.

```

4468 \int_compare:nNnT { #1 } = 1
4469 {
4470   \dim_gset:Nn \g_@@_blocks_ht_dim
4471   {
4472     \dim_max:nn
4473     \g_@@_blocks_ht_dim
4474     {
4475       \box_ht:c
4476       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4477     }
4478   }
4479   \dim_gset:Nn \g_@@_blocks_dp_dim
4480   {
4481     \dim_max:nn
4482     \g_@@_blocks_dp_dim
4483     {
4484       \box_dp:c
4485       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4486     }
4487   }
4488 }
4489 \seq_gput_right:Nx \g_@@_blocks_seq
4490 {
4491   \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_@@_pos_of_block_tl`. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of `\l_@@_pos_of_block_tl`, which is fixed by the type of current column.

```

4492 { \exp_not:n { #3 } , \l_@@_pos_of_block_tl }
4493 {
4494   \box_use_drop:c
4495   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4496 }
4497 }
4498 }

```

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.

```

4499 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
4500 {
4501   \seq_gput_right:Nx \g_@@_blocks_seq
4502   {
4503     \l_tmpa_tl
4504     { \exp_not:n { #3 } }
4505     \exp_not:n
4506     {
4507       {
4508         \bool_if:NTF \l_@@_NiceTabular_bool

```

```

4509         {
4510             \group_begin:
4511             \cs_set:Npn \arraystretch { 1 }
4512             \dim_set_eq:NN \extrarowheight \c_zero_dim
4513             #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).

```

4514             \bool_if:NT \g_@@_rotate_bool
4515             { \tl_set:Nn \l_@@_pos_of_block_tl c }
4516             \exp_args:Nnx \begin { tabular }
4517             { @ { } \l_@@_pos_of_block_tl @ { } }
4518             #5
4519             \end { tabular }
4520             \group_end:
4521         }
4522     {
4523         \group_begin:
4524         \cs_set:Npn \arraystretch { 1 }
4525         \dim_set_eq:NN \extrarowheight \c_zero_dim
4526         #4
4527         \bool_if:NT \g_@@_rotate_bool
4528         { \tl_set:Nn \l_@@_pos_of_block_tl c }
4529         \c_math_toggle_token
4530         \exp_args:Nnx \begin { array }
4531         { @ { } \l_@@_pos_of_block_tl @ { } } #5 \end { array }
4532         \c_math_toggle_token
4533         \group_end:
4534     }
4535 }
4536 }
4537 }
4538 }

```

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).

```

4539 \keys_define:nn { NiceMatrix / Block / SecondPass }
4540 {
4541     fill .tl_set:N = \l_@@_fill_tl ,
4542     fill .value_required:n = true ,
4543     draw .tl_set:N = \l_@@_draw_tl ,
4544     draw .default:n = default ,

```

The following definition for the key `color` will be deleted when we will give to the key `color` of the command `\Block` its new definition. It will be replaced by the line which is commented.

```

4545     color .code:n =
4546         \@@_error:n { Key-color-for-Block }
4547         \tl_set:Nn \l_@@_fill_tl { #1 } ,
4548     % color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
4549     color .value_required:n = true ,
4550     line-width .dim_set:N = \l_@@_line_width_dim ,
4551     line-width .value_required:n = true ,
4552     l .code:n = \tl_set:Nn \l_@@_pos_of_block_tl l ,
4553     l .value_forbidden:n = true ,
4554     r .code:n = \tl_set:Nn \l_@@_pos_of_block_tl r ,
4555     r .value_forbidden:n = true ,
4556     c .code:n = \tl_set:Nn \l_@@_pos_of_block_tl c ,
4557     c .value_forbidden:n = true ,

```

```

4558     unknown .code:n = \@@_error:n { Unknown-key-for-Block }
4559 }

```

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.

```

4560 \cs_new_protected:Npn \@@_draw_blocks:
4561 {
4562     \cs_set_eq:NN \ialign \@@_old_ialign:
4563     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
4564 }
4565 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
4566 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

4567     \int_zero_new:N \l_@@_last_row_int
4568     \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.

```

4569     \int_compare:nNnTF { #3 } > { 99 }
4570     { \int_set_eq:NN \l_@@_last_row_int \c{iRow} }
4571     { \int_set:Nn \l_@@_last_row_int { #3 } }
4572     \int_compare:nNnTF { #4 } > { 99 }
4573     { \int_set_eq:NN \l_@@_last_col_int \c{jCol} }
4574     { \int_set:Nn \l_@@_last_col_int { #4 } }
4575     \bool_lazy_or:nnTF
4576     { \int_compare_p:nNn \l_@@_last_row_int > \g_@@_row_total_int }
4577     { \int_compare_p:nNn \l_@@_last_col_int > \g_@@_col_total_int }
4578     {
4579         \int_compare:nTF
4580         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
4581         {
4582             \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
4583             \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
4584             \group_begin:
4585             \globaldefs = 1
4586             \@@_msg_redirect_name:nn { columns-not-used } { none }
4587             \group_end:
4588         }
4589         { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
4590     }
4591     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
4592 }
4593 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
4594 {

```

The sequence of the positions of the blocks will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

4595     \seq_gput_left:Nn \g_@@_pos_of_blocks_seq { { #1 } { #2 } { #3 } { #4 } }

```

The group is for the keys.

```

4596     \group_begin:
4597     \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }

```

```

4598 \bool_lazy_or:nnT
4599 { ! \tl_if_empty_p:N \l_@@_draw_tl }
4600 { \dim_compare_p:nNn \l_@@_line_width_dim > \c_zero_dim }
4601 {
4602   \tl_gput_right:Nx \g_nicematrix_code_after_tl
4603   {
4604     \@@_stroke_block:nnn
4605     { \exp_not:n { #5 } }
4606     { #1 - #2 }
4607     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
4608   }
4609   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
4610   { { #1 } { #2 } { #3 } { #4 } }
4611 }
4612 \tl_if_empty:NF \l_@@_fill_tl
4613 {
4614   \tl_gput_right:Nx \g_nicematrix_code_before_tl
4615   {
4616     \exp_not:N \rectanglecolor
4617     { \exp_not:V \l_@@_fill_tl }
4618     { #1 - #2 }
4619     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
4620   }
4621 }
4622 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
4623 {
4624   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4625   {
4626     \@@_actually_diagbox:nnnnnn
4627     { #1 }
4628     { #2 }
4629     { \int_use:N \l_@@_last_row_int }
4630     { \int_use:N \l_@@_last_col_int }
4631     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
4632   }
4633 }
4634 \hbox_set:Nn \l_@@_cell_box { #6 }
4635 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`. The latter will be used by `nicematrix` to put the label of the node. The first one won't be used explicitly.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\
& & two & \\
three & & four & five & \\
six & & seven & eight & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

| | | |
|-----------|-------|-------|
| our block | | one |
| | | two |
| three | four | five |
| six | seven | eight |

We highlight the node `1-1-block-short`

| | | |
|-----------|-------|-------|
| our block | | one |
| | | two |
| three | four | five |
| six | seven | eight |

The construction of the node corresponding to the merged cells.


```

4636 \pgfpicture
4637 \pgfrememberpicturepositiononpagetrue
4638 \pgf@relevantforpicturesizefalse
4639 \@@_qpoint:n { row - #1 }
4640 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4641 \@@_qpoint:n { col - #2 }
4642 \dim_set_eq:NN \l_tmpb_dim \pgf@x
4643 \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
4644 \dim_set_eq:NN \l_tmpc_dim \pgf@y
4645 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
4646 \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.

```

4647 \begin { pgfscope }
4648 \@@_pgf_rect_node:nnnnn
4649 { \@@_env: - #1 - #2 - block }
4650 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
4651 \end { pgfscope }

```

We construct the short node.

```

4652 \dim_set_eq:NN \l_tmpb_dim \c_max_dim
4653 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4654 {

```

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.

```

4655 \cs_if_exist:cT
4656 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
4657 {
4658 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
4659 {
4660 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
4661 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
4662 }
4663 }
4664 }

```

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.

```

4665 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
4666 {
4667 \@@_qpoint:n { col - #2 }
4668 \dim_set_eq:NN \l_tmpb_dim \pgf@x
4669 }
4670 \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
4671 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4672 {
4673 \cs_if_exist:cT
4674 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
4675 {
4676 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
4677 {
4678 \pgfpointanchor
4679 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
4680 { east }
4681 \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
4682 }
4683 }
4684 }
4685 \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
4686 {
4687 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }

```

```

4688     \dim_set_eq:NN \l_tmpd_dim \pgf@x
4689   }
4690   \@@_pgf_rect_node:nnnnn
4691     { \@@_env: - #1 - #2 - block - short }
4692     \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpe_dim

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@_pgf_rect_node:nnnnn` takes in as arguments the name of the node and two PGF points.

```

4693   \bool_if:NT \l_@@_medium_nodes_bool
4694   {
4695     \@@_pgf_rect_node:nnn
4696       { \@@_env: - #1 - #2 - block - medium }
4697       { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
4698       {
4699         \pgfpointanchor
4700           { \@@_env:
4701             - \int_use:N \l_@@_last_row_int
4702             - \int_use:N \l_@@_last_col_int - medium
4703           }
4704           { south-east }
4705       }
4706   }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

4707   \int_compare:nNnTF { #1 } = { #3 }
4708   {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

4709     \int_compare:nNnTF { #1 } = 0
4710     { \l_@@_code_for_first_row_tl }
4711     {
4712       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4713       { \l_@@_code_for_last_row_tl }
4714     }

```

If the block has only one row, we want the label of the block perfectly aligned on the baseline of the row. That’s why we have constructed a `\pgfcoordinate` on the baseline of the row, in the first column of the array. Now, we retrieve the y -value of that node and we store it in `\l_tmpa_dim`.

```

4715     \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

4716     \pgfpointanchor
4717     { \@@_env: - #1 - #2 - block - short }
4718     {
4719       \str_case:Vn \l_@@_pos_of_block_tl
4720       {
4721         c { center }
4722         l { west }
4723         r { east }
4724       }
4725     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

4726     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
4727     \pgfset { inner-sep = \c_zero_dim }
4728     \pgfnode
4729     { rectangle }
4730     {
4731       \str_case:Vn \l_@@_pos_of_block_tl
4732       {
4733         c { base }
4734         l { base~west }
4735         r { base~east }
4736       }
4737     }

```

```

4738         { \box_use_drop:N \l_@@_cell_box } { } { }
4739     }

```

If the number of rows is different of 1, we will put the label of the block by using the short node (the label of the block has been composed in `\l_@@_cell_box`).

```

4740     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

4741         \int_compare:nNnT { #2 } = 0
4742         { \tl_set:Nn \l_@@_pos_of_block_tl r }
4743         \bool_if:nT \g_@@_last_col_found_bool
4744         {
4745             \int_compare:nNnT { #2 } = \g_@@_col_total_int
4746             { \tl_set:Nn \l_@@_pos_of_block_tl 1 }
4747         }
4748         \pgftransformshift
4749         {
4750             \pgfpointanchor
4751             { \@@_env: - #1 - #2 - block - short }
4752             {
4753                 \str_case:Nn \l_@@_pos_of_block_tl
4754                 {
4755                     c { center }
4756                     l { west }
4757                     r { east }
4758                 }
4759             }
4760         }
4761         \pgfset { inner~sep = \c_zero_dim }
4762         \pgfnode
4763         { rectangle }
4764         {
4765             \str_case:Nn \l_@@_pos_of_block_tl
4766             {
4767                 c { center }
4768                 l { west }
4769                 r { east }
4770             }
4771         }
4772         { \box_use_drop:N \l_@@_cell_box } { } { }
4773     }
4774 \endpgfpicture
4775 \group_end:
4776 }

```

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).

```

4777 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
4778 {
4779     \group_begin:
4780     \tl_clear:N \l_@@_draw_tl
4781     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
4782     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
4783     \pgfpicture
4784     \pgfrememberpicturerepositiononpagetrue
4785     \pgf@relevantforpicturesizefalse
4786     \tl_if_empty:NF \l_@@_draw_tl
4787     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

4788         \str_if_eq:NnTF \l_@@_draw_tl { default }

```

```

4789         { \CT@arc@ }
4790         { \pgfsetstrokecolor { \l_@@_draw_tl } }
4791     }
4792     \@@_cut_on_hyphen:w #2 \q_stop
4793     \bool_lazy_and:nnT
4794     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
4795     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
4796     {
4797         \@@_qpoint:n { row - \l_tmpa_tl }
4798         \dim_set:Nn \l_tmpb_dim { \pgf@y }
4799         \@@_qpoint:n { col - \l_tmpb_tl }
4800         \dim_set:Nn \l_tmpc_dim { \pgf@x }
4801         \@@_cut_on_hyphen:w #3 \q_stop
4802         \int_compare:nNnT \l_tmpa_tl > \c@iRow
4803         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
4804         \int_compare:nNnT \l_tmpb_tl > \c@jCol
4805         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4806         \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
4807         \dim_set:Nn \l_tmpa_dim { \pgf@y }
4808         \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4809         \dim_set:Nn \l_tmpd_dim { \pgf@x }
4810         \pgfpathrectanglecorners
4811         { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4812         { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
4813         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
4814         \pgfusepathqstroke
4815     }
4816     \endpgfpicture
4817     \group_end:
4818 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

4819 \keys_define:nn { NiceMatrix / BlockStroke }
4820 {

```

We will uncomment the following line when we will give to the key `color` of the command `\Block` its new definition.

```

4821     % color .tl_set:N = \l_@@_draw_tl ,
4822     draw .tl_set:N = \l_@@_draw_tl ,
4823     draw .default:n = default ,
4824     line-width .dim_set:N = \l_@@_line_width_dim ,
4825 }

```

How to draw the dotted lines transparently

```

4826 \cs_set_protected:Npn \@@_renew_matrix:
4827 {
4828     \RenewDocumentEnvironment { pmatrix } { } {
4829         { \pNiceMatrix }
4830         { \endpNiceMatrix }
4831     }
4832     \RenewDocumentEnvironment { vmatrix } { } {
4833         { \vNiceMatrix }
4834         { \endvNiceMatrix }
4835     }
4836     \RenewDocumentEnvironment { Vmatrix } { } {
4837         { \VNiceMatrix }
4838         { \endVNiceMatrix }
4839     }
4840     \RenewDocumentEnvironment { bmatrix } { } {
4841         { \bNiceMatrix }
4842         { \endbNiceMatrix }
4843     }
4844     \RenewDocumentEnvironment { Bmatrix } { } {
4845         { \BNiceMatrix }
4846         { \endBNiceMatrix }
4847     }

```

```
4843 }
```

Automatic arrays

```
4844 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
4845 {
4846   \int_set:Nn \l_@@_nb_rows_int { #1 }
4847   \int_set:Nn \l_@@_nb_cols_int { #2 }
4848 }

4849 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m 0 { } m 0 { } m ! 0 { } }
4850 {
4851   \int_zero_new:N \l_@@_nb_rows_int
4852   \int_zero_new:N \l_@@_nb_cols_int
4853   \@@_set_size:n #4 \q_stop
4854   \begin { NiceArrayWithDelims } { #1 } { #2 }
4855     { * { \l_@@_nb_cols_int } { c } } [ #3 , #5 , #7 ]
4856   \int_compare:nNnT \l_@@_first_row_int = 0
4857     {
4858     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
4859     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
4860     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
4861   }
4862   \prg_replicate:nn \l_@@_nb_rows_int
4863   {
4864     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
```

You put { } before #6 to avoid a hasty expansion of a potential `\arabic{iRow}` at the beginning of the row which would result in an incorrect value of that `iRow` (since `iRow` is incremented in the first cell of the row of the `\halign`).

```
4865   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
4866   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
4867 }
4868 \int_compare:nNnT \l_@@_last_row_int > { -2 }
4869 {
4870   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
4871   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
4872   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
4873 }
4874 \end { NiceArrayWithDelims }
4875 }

4876 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
4877 {
4878   \cs_set_protected:cpn { #1 AutoNiceMatrix }
4879   {
4880     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
4881     \AutoNiceMatrixWithDelims { #2 } { #3 }
4882   }
4883 }

4884 \@@_define_com:nnn p ( )
4885 \@@_define_com:nnn b [ ]
4886 \@@_define_com:nnn v | |
4887 \@@_define_com:nnn V \ | \ |
4888 \@@_define_com:nnn B \{ \}
```

We define also an command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```
4889 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
4890 {
4891   \group_begin:
4892     \bool_set_true:N \l_@@_NiceArray_bool
4893     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
4894   \group_end:
4895 }
```

The redefinition of the command `\dotfill`

```

4896 \cs_set_eq:NN \@@_old_dotfill \dotfill
4897 \cs_new_protected:Npn \@@_dotfill:
4898 {

```

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}`).

```

4899   \@@_old_dotfill
4900   \bool_if:NT \l_@@_NiceTabular_bool
4901     { \group_insert_after:N \@@_dotfill_ii: }
4902     { \group_insert_after:N \@@_dotfill_i: }
4903   }
4904 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
4905 \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`.

```

4906 \cs_new_protected:Npn \@@_dotfill_iii:
4907 { \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`.

```

4908 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
4909 {
4910   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4911   {
4912     \@@_actually_diagbox:nnnnnn
4913     { \int_use:N \c@iRow }
4914     { \int_use:N \c@jCol }
4915     { \int_use:N \c@iRow }
4916     { \int_use:N \c@jCol }
4917     { \exp_not:n { #1 } }
4918     { \exp_not:n { #2 } }
4919   }

```

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 `except-corners`.

```

4920   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
4921   {
4922     { \int_use:N \c@iRow }
4923     { \int_use:N \c@jCol }
4924     { \int_use:N \c@iRow }
4925     { \int_use:N \c@jCol }
4926   }
4927 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it’s possible to use `\diagbox` in a `\Block`). The two other are the elements to draw below and above the diagonal line.

```

4928 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
4929 {
4930   \pgfpicture
4931   \pgf@relevantforpicturesizefalse
4932   \pgfrememberpicturepositiononpagetrue
4933   \@@_qpoint:n { row - #1 }
4934   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4935   \@@_qpoint:n { col - #2 }
4936   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4937   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }

```

```

4938 \@@_qpoint:n { row - \@@_succ:n { #3 } }
4939 \dim_set_eq:NN \l_tmpc_dim \pgf@y
4940 \@@_qpoint:n { col - \@@_succ:n { #4 } }
4941 \dim_set_eq:NN \l_tmpd_dim \pgf@x
4942 \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4943 {

```

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.

```

4944 \CT@arc@
4945 \pgfsetroundcap
4946 \pgfusepathqstroke
4947 }
4948 \pgfset { inner-sep = 1 pt }
4949 \pgfscope
4950 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4951 \pgfnode { rectangle } { south-west }
4952 { \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }
4953 \endpgfscope
4954 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
4955 \pgfnode { rectangle } { north-east }
4956 { \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
4957 \endpgfpicture
4958 }

```

The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 89.

In the environments of `nicematrix`, `\CodeAfter` will be linked to the following command `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

4959 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_i:n }

```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_i:n` which do *not* begin with `\omit` (and thus, the user will be able to use `\CodeAfter` without error and without the need to prefix by `\omit`).

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

4960 \cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
4961 {
4962 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
4963 \@@_CodeAfter_ii:n
4964 }

```

We catch the argument of the command `\end` (in `#1`).

```

4965 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1
4966 {

```

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.

```

4967 \str_if_eq:eeTF \@currenvir { #1 }
4968 { \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`.

```

4969 {
4970 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
4971 \@@_CodeAfter_i:n
4972 }
4973 }

```

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.

```
4974 \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.

```
4975 \bool_new:N \c_@@_footnote_bool
```

```
4976 \@@_msg_new:nnn { Unknown-option-for-package }
```

```
4977 {
```

```
4978   The~key~'\l_keys_key_str'~is-unknown.  \\  
4979   If~you~go~on,~it~will~be~ignored.  \\  
4980   For~a~list~of~the~available~keys,~type~H~<return>.
```

```
4981 }
```

```
4982 {
```

```
4983   The~available~keys~are~(in~alphabetic~order):~  
4984   define-L-C-R,~  
4985   footnote,~  
4986   footnotehyper,~  
4987   renew-dots,~  
4988   renew-matrix~and~  
4989   transparent.
```

```
4990 }
```

```
4991 \keys_define:nn { NiceMatrix / Package }
```

```
4992 {
```

```
4993   define-L-C-R .bool_set:N = \c_@@_define_L_C_R_bool ,  
4994   define-L-C-R .default:n = true ,  
4995   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,  
4996   renew-dots .value_forbidden:n = true ,  
4997   renew-matrix .code:n = \@@_renew_matrix: ,  
4998   renew-matrix .value_forbidden:n = true ,  
4999   transparent .meta:n = { renew-dots , renew-matrix } ,  
5000   transparent .value_forbidden:n = true ,  
5001   footnote .bool_set:N = \c_@@_footnote_bool ,  
5002   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,  
5003   unknown .code:n = \@@_error:n { Unknown-option-for-package }
```

```
5004 }
```

```
5005 \ProcessKeysOptions { NiceMatrix / Package }
```

```
5006
```

```
5007 \@@_msg_new:nn { footnote-with-footnotehyper-package }
```

```
5008 {
```

```
5009   You~can't~use~the~option~'footnote'~because~the~package~  
5010   footnotehyper~has~already~been~loaded.~  
5011   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~  
5012   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~  
5013   of~the~package~footnotehyper.\\  
5014   If~you~go~on,~the~package~footnote~won't~be~loaded.
```

```
5015 }
```

```
5016 \@@_msg_new:nn { footnotehyper-with-footnote-package }
```

```
5017 {
```

```
5018   You~can't~use~the~option~'footnotehyper'~because~the~package~  
5019   footnote~has~already~been~loaded.~  
5020   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~  
5021   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~  
5022   of~the~package~footnote.\\
```

```
5023 }
```



```

5022   If~you-go-on,~the~package~footnotehyper~won't~be~loaded.
5023 }

5024 \bool_if:NT \c_@@_footnote_bool
5025 {
5026   \@ifclassloaded { beamer }
5027   { \msg_info:nn { nicematrix } { Option~incompatible-with-Beamer } }
5028   {
5029     \@ifpackageloaded { footnotehyper }
5030     { \@@_error:n { footnote~with~footnotehyper~package } }
5031     { \usepackage { footnote } }
5032   }
5033 }

5034 \bool_if:NT \c_@@_footnotehyper_bool
5035 {
5036   \@ifclassloaded { beamer }
5037   { \@@_info:n { Option~incompatible-with-Beamer } }
5038   {
5039     \@ifpackageloaded { footnote }
5040     { \@@_error:n { footnotehyper~with~footnote~package } }
5041     { \usepackage { footnotehyper } }
5042   }
5043   \bool_set_true:N \c_@@_footnote_bool
5044 }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

Error messages of the package

```

5045 \seq_new:N \c_@@_types_of_matrix_seq
5046 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
5047 {
5048   NiceMatrix ,
5049   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
5050 }
5051 \seq_set_map_x:NNn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
5052 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is executed. This command raises an error but try to give the best information to the user in the error message. The command `\seq_if_in:NVTf` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

5053 \cs_new_protected:Npn \@@_error_too_much_cols:
5054 {
5055   \seq_if_in:NVTf \c_@@_types_of_matrix_seq \g_@@_name_env_str
5056   {
5057     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
5058     { \@@_fatal:n { too~much~cols~for~matrix } }
5059     {
5060       \bool_if:Nf \l_@@_last_col_without_value_bool
5061       { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
5062     }
5063   }
5064   { \@@_fatal:n { too~much~cols~for~array } }
5065 }

```

The following command must *not* be protected since it's used in an error message.

```

5066 \cs_new:Npn \@@_message_hdotsfor:
5067 {
5068   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl

```

```

5069     { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
5070   }
5071 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
5072   {
5073     You~try~to~use~more~columns~than~allowed~by~your~
5074     \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
5075     columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
5076     exterior~columns).~This~error~is~fatal.
5077   }
5078 \@@_msg_new:nn { too-much-cols-for-matrix }
5079   {
5080     You~try~to~use~more~columns~than~allowed~by~your~
5081     \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
5082     number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
5083     'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
5084     This~error~is~fatal.
5085   }

```

For the following message, remind that the test is not done after the construction of the array but in each row. That's why we have to put `\c@jCol-1` and not `\c@jCol`.

```

5086 \@@_msg_new:nn { too-much-cols-for-array }
5087   {
5088     You~try~to~use~more~columns~than~allowed~by~your~
5089     \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
5090     \int_use:N \g_@@_static_num_of_col_int\
5091     ~(plus~the~potential~exterior~ones).~
5092     This~error~is~fatal.
5093   }
5094 \@@_msg_new:nn { last-col-not-used }
5095   {
5096     The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
5097     in~your~\@@_full_name_env:~.~However,~you~can~go~on.
5098   }
5099 \@@_msg_new:nn { columns-not-used }
5100   {
5101     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
5102     \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\
5103     You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
5104   }
5105 \@@_msg_new:nn { in-first-col }
5106   {
5107     You~can't~use~the~command~\#1 in~the~first~column~(number~0)~of~the~array.\
5108     If~you~go~on,~this~command~will~be~ignored.
5109   }
5110 \@@_msg_new:nn { in-last-col }
5111   {
5112     You~can't~use~the~command~\#1 in~the~last~column~(exterior)~of~the~array.\
5113     If~you~go~on,~this~command~will~be~ignored.
5114   }
5115 \@@_msg_new:nn { in-first-row }
5116   {
5117     You~can't~use~the~command~\#1 in~the~first~row~(number~0)~of~the~array.\
5118     If~you~go~on,~this~command~will~be~ignored.
5119   }
5120 \@@_msg_new:nn { in-last-row }
5121   {
5122     You~can't~use~the~command~\#1 in~the~last~row~(exterior)~of~the~array.\
5123     If~you~go~on,~this~command~will~be~ignored.
5124   }
5125 \@@_msg_new:nn { bad-option-for-line-style }

```

```

5126 {
5127     Since-you-haven't-loaded-Tikz,~the-only-value-you-can-give-to~'line-style'~
5128     is~'standard'.~If-you-go-on,~this-key-will-be-ignored.
5129 }

5130 \@@_msg_new:nn { Unknown-option-for~xdots }
5131 {
5132     As-for~now~there-is-only-three-key-available-here:~'color',~'line-style'~
5133     and~'shorten'~(and-you-try-to-use~'\l_keys_key_str').~If-you-go-on,~
5134     this-key-will-be-ignored.
5135 }

5136 \@@_msg_new:nn { Unknown-option-for~rowcolors }
5137 {
5138     As-for~now~there-is-only-two-keys-available-here:~'cols'~and~'respect-blocks'~
5139     (and-you-try-to-use~'\l_keys_key_str').~If-you-go-on,~
5140     this-key-will-be-ignored.
5141 }

5142 \@@_msg_new:nn { Key~color~for~Block }
5143 {
5144     The-key~'color'~for-the-command~\token_to_str:N \Block\
5145     is-deprecated:~you-should-use~'fill'~instead.\\
5146     You-can-go-on-for-this-time-but-remember-that-that-key~
5147     will-be-deleted-in-a-future-version.
5148 }

5149 \@@_msg_new:nn { ampersand~in~light-syntax }
5150 {
5151     You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
5152     you-have-used-the-key~'light-syntax'.~This-error-is-fatal.
5153 }

5154 \@@_msg_new:nn { double-backslash~in~light-syntax }
5155 {
5156     You-can't-use~\token_to_str:N \\~to-separate-rows-because-you-have-used~
5157     the-key~'light-syntax'.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
5158     (set-by-the-key~'end-of-row').~This-error-is-fatal.
5159 }

5160 \@@_msg_new:nn { standard-cline~in~document }
5161 {
5162     The-key~'standard-cline'~is-available-only-in-the-preamble.\\
5163     If-you-go-on-this-command-will-be-ignored.
5164 }

5165 \@@_msg_new:nn { old-column-type }
5166 {
5167     The-column-type~'#1'~is-no-longer-defined-in~'nicematrix'.~
5168     Since-version-5.0,~you-have-to-use~'l',~'c'~and~'r'~instead-of~'L',~
5169     'C'~and~'R'.~You-can-also-use-the-key~'define-L-C-R'.\\
5170     This-error-is-fatal.
5171 }

5172 \@@_msg_new:nn { bad-value~for~baseline }
5173 {
5174     The-value-given-to~'baseline'~(\int_use:N \l_tmpa_int)~is-not~
5175     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
5176     \int_use:N \g_@@_row_total_int\ or-equal-to~'t',~'c'~or~'b'.\\
5177     If-you-go-on,~a-value-of-1~will-be-used.
5178 }

5179 \@@_msg_new:nn { empty~environment }
5180 { Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal. }

5181 \@@_msg_new:nn { unknown~cell~for~line~in~code~after }
5182 {
5183     Your-command~\token_to_str:N\line\{#1\}\{#2\}~in-the~'code-after'~
5184     can't-be-executed-because-a-cell-doesn't-exist.\\

```

```

5185     If-you-go-on-this-command-will-be-ignored.
5186 }

5187 \@@_msg_new:nn { Hdotsfor~in~col~0 }
5188 {
5189     You-can't-use~\token_to_str:N \Hdotsfor\ in-an-exterior~column-of~
5190     the~array.~This-error-is-fatal.
5191 }

5192 \@@_msg_new:nn { bad~corner }
5193 {
5194     #1-is-an-incorrect-specification-for-a-corner~(in-the-keys~
5195     'except-corners'~and~'hvlines-except-corners').~The-available~
5196     values-are:~NW,~SW,~NE-and~SE.\\
5197     If-you-go-on,~this-specification-of~corner-will-be-ignored.
5198 }

5199 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
5200 {
5201     In-the~\@@_full_name_env:,~you-must-use~the-key~
5202     'last-col'~without~value.\\
5203     However,~you-can-go-on~for~this-time~
5204     (the-value~'\l_keys_value_tl'~will-be-ignored).
5205 }

5206 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
5207 {
5208     In~\NiceMatrixoptions,~you-must-use~the-key~
5209     'last-col'~without~value.\\
5210     However,~you-can-go-on~for~this-time~
5211     (the-value~'\l_keys_value_tl'~will-be-ignored).
5212 }

5213 \@@_msg_new:nn { Block~too~large~1 }
5214 {
5215     You-try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
5216     too~small~for~that~block. \\
5217 }

5218 \@@_msg_new:nn { Block~too~large~2 }
5219 {
5220     The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
5221     \g_@@_static_num_of_col_int\
5222     columns~but~you-use-only~\int_use:N \c@jCol\ and~that's~why~a~block~
5223     specified~in~the~cell~#1-#2~can't~be~drawn.~You-should~add~some~ampersands~
5224     (&)~at~the~end~of~the~first~row~of~your~
5225     \@@_full_name_env:.\\
5226     If-you-go-on,~this~block~and~maybe~others~will~be~ignored.
5227 }

5228 \@@_msg_new:nn { unknown~column~type }
5229 {
5230     The~column~type~'#1'~in~your~\@@_full_name_env:\
5231     is~unknown. \\
5232     This-error-is-fatal.
5233 }

5234 \@@_msg_new:nn { tabularnote~forbidden }
5235 {
5236     You-can't-use~the~command~\token_to_str:N\ tabularnote\
5237     ~in~a~\@@_full_name_env:.~This-command-is~available~only~in~
5238     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
5239     If-you-go-on,~this~command~will~be~ignored.
5240 }

5241 \@@_msg_new:nn { bottomule~without~booktabs }
5242 {
5243     You-can't-use~the~key~'tabular/bottomrule'~because~you~haven't~
5244     loaded~'booktabs'.\\

```

```

5245     If~you~go~on,~this~key~will~be~ignored.
5246 }
5247 \@@_msg_new:nn { enumitem~not~loaded }
5248 {
5249     You~can't~use~the~command~\token_to_str:N\tabularnote\
5250     ~because~you~haven't~loaded~'enumitem'.\\
5251     If~you~go~on,~this~command~will~be~ignored.
5252 }
5253 \@@_msg_new:nn { Wrong~last~row }
5254 {
5255     You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
5256     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
5257     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
5258     last-row.~You~can~avoid~this~problem~by~using~'last-row'~
5259     without~value~(more~compilations~might~be~necessary).
5260 }
5261 \@@_msg_new:nn { Yet~in~env }
5262 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }
5263 \@@_msg_new:nn { Outside~math~mode }
5264 {
5265     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
5266     (and~not~in~\token_to_str:N \vcenter).\\
5267     This~error~is~fatal.
5268 }
5269 \@@_msg_new:nn { Bad~value~for~letter~for~dotted~lines }
5270 {
5271     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
5272     If~you~go~on,~it~will~be~ignored.
5273 }
5274 \@@_msg_new:nnn { Unknown~key~for~Block }
5275 {
5276     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
5277     \Block.\\ If~you~go~on,~it~will~be~ignored. \\
5278     For~a~list~of~the~available~keys,~type~H~<return>.
5279 }
5280 {
5281     The~available~keys~are~(in~alphabetic~order):~,~c,~draw,~fill,~l,~
5282     line-width~and~r.
5283 }
5284 \@@_msg_new:nnn { Unknown~key~for~notes }
5285 {
5286     The~key~'\l_keys_key_str'~is~unknown.\\
5287     If~you~go~on,~it~will~be~ignored. \\
5288     For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
5289 }
5290 {
5291     The~available~keys~are~(in~alphabetic~order):~
5292     bottomrule,~
5293     code-after,~
5294     code-before,~
5295     enumitem-keys,~
5296     enumitem-keys-para,~
5297     para,~
5298     label-in-list,~
5299     label-in-tabular~and~
5300     style.
5301 }
5302 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
5303 {
5304     The~key~'\l_keys_key_str'~is~unknown~for~the~command~

```

```

5305 \token_to_str:N \NiceMatrixOptions. \\
5306 If~you~go~on,~it~will~be~ignored. \\
5307 For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
5308 }
5309 {
5310 The~available~keys~are~(in~alphabetic~order):~
5311 allow-duplicate-names,~
5312 cell-space-bottom-limit,~
5313 cell-space-limits,~
5314 cell-space-top-limit,~
5315 code-for-first-col,~
5316 code-for-first-row,~
5317 code-for-last-col,~
5318 code-for-last-row,~
5319 create-extra-nodes,~
5320 create-medium-nodes,~
5321 create-large-nodes,~
5322 delimiters-color,~
5323 end-of-row,~
5324 first-col,~
5325 first-row,~
5326 hlines,~
5327 hvlines,~
5328 hvlines-except-corners,~
5329 last-col,~
5330 last-row,~
5331 left-margin,~
5332 letter-for-dotted-lines,~
5333 light-syntax,~
5334 notes~(several subkeys),~
5335 nullify-dots,~
5336 renew-dots,~
5337 renew-matrix,~
5338 right-margin,~
5339 small,~
5340 transparent,~
5341 vlines,~
5342 xdots/color,~
5343 xdots/shorten~and~
5344 xdots/line-style.
5345 }
5346 \@@_msg_new:nnn { Unknown~option~for~NiceArray }
5347 {
5348 The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
5349 \{NiceArray\}. \\
5350 If~you~go~on,~it~will~be~ignored. \\
5351 For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
5352 }
5353 {
5354 The~available~keys~are~(in~alphabetic~order):~
5355 b,~
5356 baseline,~
5357 c,~
5358 cell-space-bottom-limit,~
5359 cell-space-limits,~
5360 cell-space-top-limit,~
5361 code-after,~
5362 code-for-first-col,~
5363 code-for-first-row,~
5364 code-for-last-col,~
5365 code-for-last-row,~
5366 colortbl-like,~
5367 columns-width,~

```

```

5368 create-extra-nodes,~
5369 create-medium-nodes,~
5370 create-large-nodes,~
5371 delimiters-color,~
5372 extra-left-margin,~
5373 extra-right-margin,~
5374 first-col,~
5375 first-row,~
5376 hlines,~
5377 hvlines,~
5378 hvlines-except-corners,~
5379 last-col,~
5380 last-row,~
5381 left-margin,~
5382 light-syntax,~
5383 name,~
5384 notes/bottomrule,~
5385 notes/para,~
5386 nullify-dots,~
5387 renew-dots,~
5388 right-margin,~
5389 rules/color,~
5390 rules/width,~
5391 small,~
5392 t,~
5393 vlines,~
5394 xdots/color,~
5395 xdots/shorten-and~
5396 xdots/line-style.
5397 }

```

This error message is used for the set of keys NiceMatrix/NiceMatrix and NiceMatrix/pNiceArray (but not by NiceMatrix/NiceArray because, for this set of keys, there is also the keys t, c and b).

```

5398 \@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
5399 {
5400   The~key~'\l_keys_key_str'~is~unknown~for~the~
5401   \@@_full_name_env:. \\\
5402   If~you~go~on,~it~will~be~ignored. \\\
5403   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
5404 }
5405 {
5406   The~available~keys~are~(in~alphabetic~order):~
5407   b,~
5408   baseline,~
5409   c,~
5410   cell-space-bottom-limit,~
5411   cell-space-limits,~
5412   cell-space-top-limit,~
5413   code-after,~
5414   code-for-first-col,~
5415   code-for-first-row,~
5416   code-for-last-col,~
5417   code-for-last-row,~
5418   colortbl-like,~
5419   columns-width,~
5420   create-extra-nodes,~
5421   create-medium-nodes,~
5422   create-large-nodes,~
5423   delimiters-color,~
5424   extra-left-margin,~
5425   extra-right-margin,~
5426   first-col,~
5427   first-row,~
5428   hlines,~

```

```

5429     hvlines,~
5430     hvlines-except-corners,~
5431     l,~
5432     last-col,~
5433     last-row,~
5434     left-margin,~
5435     light-syntax,~
5436     name,~
5437     nullify-dots,~
5438     r,~
5439     renew-dots,~
5440     right-margin,~
5441     rules/color,~
5442     rules/width,~
5443     small,~
5444     t,~
5445     vlines,~
5446     xdots/color,~
5447     xdots/shorten~and~
5448     xdots/line-style.
5449 }
5450 \@@_msg_new:nnn { Unknown~option~for~NiceTabular }
5451 {
5452   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
5453   \{NiceTabular\}. \\
5454   If~you~go~on,~it~will~be~ignored. \\
5455   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
5456 }
5457 {
5458   The~available~keys~are~(in~alphabetic~order):~
5459   b,~
5460   baseline,~
5461   c,~
5462   cell-space-bottom-limit,~
5463   cell-space-limits,~
5464   cell-space-top-limit,~
5465   code-after,~
5466   code-for-first-col,~
5467   code-for-first-row,~
5468   code-for-last-col,~
5469   code-for-last-row,~
5470   colortbl-like,~
5471   columns-width,~
5472   create-extra-nodes,~
5473   create-medium-nodes,~
5474   create-large-nodes,~
5475   extra-left-margin,~
5476   extra-right-margin,~
5477   first-col,~
5478   first-row,~
5479   hlines,~
5480   hvlines,~
5481   hvlines-except-corners,~
5482   last-col,~
5483   last-row,~
5484   left-margin,~
5485   light-syntax,~
5486   name,~
5487   notes/bottomrule,~
5488   notes/para,~
5489   nullify-dots,~
5490   renew-dots,~
5491   right-margin,~

```



```

5492     rules/color,~
5493     rules/width,~
5494     t,~
5495     vlines,~
5496     xdots/color,~
5497     xdots/shorten~and~
5498     xdots/line-style.
5499 }

5500 \@@_msg_new:nnn { Duplicate-name }
5501 {
5502     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
5503     the~same~environment~name~twice.~You~can~go~on,~but,~
5504     maybe,~you~will~have~incorrect~results~especially~
5505     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
5506     message~again,~use~the~key~'allow-duplicate-names'~in~
5507     '\token_to_str:N \NiceMatrixOptions'.\\
5508     For~a~list~of~the~names~already~used,~type~H~<return>. \\
5509 }
5510 {
5511     The~names~already~defined~in~this~document~are:~
5512     \seq_use:Nnnn \g_@@_names_seq { ,~ } { ,~ } { ~and~ }.
5513 }

5514 \@@_msg_new:nn { Option-auto-for-columns-width }
5515 {
5516     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
5517     If~you~go~on,~the~key~will~be~ignored.
5518 }

```

18 History

The successive versions of the file `nicematrix.sty` provided by TeXLive are available on the SVN server of TeXLive:

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

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types L, C and R.

Changes between version 1.2 and 1.3

New environment `{pNiceArrayC}` and its variants.

Correction of a bug in the definition of `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` (in fact, it was a typo).

Options are now available locally in `{pNiceMatrix}` and its variants.

The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types `w` and `W` can now be used in the environments `{NiceArray}`, `{pNiceArrayC}` and its variants with the same meaning as in the package `array`.

New option `columns-width` to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of `nicematrix` were focused on the continuous dotted lines whereas the version 2.0 of `nicematrix` provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment `{pNiceArrayRC}`. With this new implementation, there is no restriction on the width of the columns.

The package `nicematrix` no longer loads `mathtools` but only `amsmath`.

Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option `code-for-first-row` is now available in the command `\NiceMatrixOptions`.

Following a discussion on TeX StackExchange⁴⁹, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁵⁰

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a & \cdots \\ 0 & & 0 \end{pmatrix} L_i$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

⁴⁹cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁵⁰Before this version, there was an error when using `nicematrix` with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by `nicematrix` because they use the options `overlay` and `remember picture`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`).

Possibility to draw vertical dotted lines to separate columns with the specifier “:” in the preamble (similar to the classical specifier “|” and the specifier “:” of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “:” in the preamble.

Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.

Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of “|”) as `\hdotsfor` does.

Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.

Error message when the user gives an incorrect value for `last-row`.

A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol “:” (in the preamble of the array) and `\line` in `code-after`).

The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.

The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by `colortbl`.

Correction of a bug: it was not possible to use the colon “:” in the preamble of an array when `pdflatex` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

The option `columns-width=auto` doesn’t need any more a second compilation.

The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).

The previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange⁵¹, optimization of Tikz externalization is disabled in the environments of `nicematrix` when the class `standalone` or the package `standalone` is used.

⁵¹cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

Changes between version 3.5 and 3.6

LaTeX counters `iRow` and `jCol` available in the cells of the array.

Addition of `\normalbaselines` before the construction of the array: in environments like `{align}` of `amsmath` the value of `\baselineskip` is changed and if the options `first-row` and `last-row` were used in an environment of `nicematrix`, the position of the delimiters was wrong.

A warning is written in the `.log` file if an obsolete environment is used.

There is no longer artificial errors `Duplicate-name` in the environments of `amsmath`.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.

New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programming for the command `\Block` when the block has only one row. With this programming, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with `\multicolumn`. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands `\NiceMatrixLastEnv` and `\OnlyMainNiceMatrix`.

New options `create-medium-nodes` and `create-large-nodes`.

Changes between version 3.9 and 3.10

New option `light-syntax` (and `end-of-row`).

New option `dotted-lines-margin` for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

Correction of a bug linked to `first-row` and `last-row`.

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -block and, if the creation of the “medium nodes” is required, a node $i-j$ -block-medium is created.

If the user tries to use more columns than allowed by its environment, an error is raised by `nicematrix` (instead of a low-level error).

The package must be loaded with the option `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the “last row”.

The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customization of the dotted lines.

In the environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, etc.), it’s possible to use the options `l` (`=L`) or `r` (`=R`) to specify the type of the columns.

The starred versions of the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots` and `\Iddots` are deprecated since the version 3.1 of `nicematrix`. Now, one should load `nicematrix` with the option `starred-commands` to avoid an error at the compilation.

The code of `nicematrix` no longer uses `Tikz` but only `PGF`. By default, `Tikz` is *not* loaded by `nicematrix`.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on [stackoverflow](#)).

Better error messages when the user uses `&` or `\\` when `light-syntax` is in force.

Changes between versions 3.14 and 3.15

It’s possible to put labels on the dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and the command `\line` in the `code-after` with the tokens `_` and `^`.

The option `baseline` is now available in all the environments of `nicematrix`. Before, it was available only in `{NiceArray}`.

New keyword `\CodeAfter` (in the environments of `nicematrix`).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`

Commands to color cells, rows and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`

New command `\diagbox`

The key `hvline` don’t draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It’s now possible to write `\begin{pNiceMatrix}a&b\\c&d\end{pNiceMatrix}^2` with the expected result.

Changes between versions 4.2 and 4.3

The horizontal centering of the content of a `\Block` is correct even when an instruction such as `!\quad` is used in the preamble of the array.

It’s now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.

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

Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).

Environment `{NiceTabular*}`

Command `\Vdotsfor` similar to `\Hdotsfor`

The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.

Key `respect-blocks` for `\rowcolors` (with a `s`) in the `code-before`.

The variable `\g_nicematrix_code_before_tl` is now public.

The key `baseline` can take in as value of the form `line-i` to align the `\hline` in the row `i`.

The key `hvlines-except-corners` may take in as value a list of corners (eg: `NW,SE`).

Changes between versions 5.2 and 5.3

Keys `c`, `r` and `l` for the command `\Block`.

It's possible to use the key `draw-first` with `\Ddots` and `\Iddots` to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key `tabularnote`.

Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put `\omit` before `\CodeAfter`.

Correction of a bug: the tabular notes `\tabularnotes` were not composed when present in a block (except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.

New command `\NotEmpty`.

Changes between versions 5.6 and 5.7

New key `delimiters-color`

Keys `fill`, `draw` and `line-width` for the command `\Block`.

Changes between versions 5.7 and 5.8

Keys `cols` and `restart` of the command `\rowcolors` in the `code-before`.

Modification of the behaviour of `\\` in the columns of type `p`, `m` or `b` (for a behaviour similar to the environments of `array`).

Better error messages for the command `\Block`.

Changes between versions 5.8 and 5.9

Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.

New key `cell-space-limits`.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

| Symbols | |
|--|---|
| @@ commands: | |
| <code>\@@_Block:</code> | 1083, 4364 |
| <code>\@@_Block_i</code> | 4366, 4369 |
| <code>\@@_Block_ii:nnnnn</code> | 4369, 4370 |
| <code>\@@_Block_iv:nnnnn</code> | 4400, 4404 |
| <code>\@@_Block_iv:nnnnnn</code> | 4563, 4565 |
| <code>\@@_Block_v:nnnnn</code> | 4401, 4499 |
| <code>\@@_Block_v:nnnnnn</code> | 4591, 4593 |
| <code>\@@_Cdots</code> | 1008, 1073, 2978 |
| <code>\g_@@_Cdots_lines_tl</code> | 1100, 2318 |
| <code>\@@_Cell:</code> | 194, 770, 1478, 1525, 1543, 2104, 3078, 3079, 3080, 3081, 3082, 3083, 3084, 3085, 3086, 3087, 3088, 3089 |
| <code>\@@_CodeAfter:</code> | 1087, 4959 |
| <code>\@@_CodeAfter_i:n</code> | 772, 4959, 4960, 4971 |
| <code>\@@_CodeAfter_ii:n</code> | 4963, 4965 |
| <code>\@@_Ddots</code> | 1010, 1075, 3008 |
| <code>\g_@@_Ddots_lines_tl</code> | 1103, 2316 |
| <code>\g_@@_HVDotsfor_lines_tl</code> | 1105, 2314, 3137, 3213, 5068 |
| <code>\@@_Hdotsfor:</code> | 1013, 1080, 3113 |
| <code>\@@_Hdotsfor:nnnn</code> | 3139, 3151 |
| <code>\@@_Hdotsfor_i</code> | 3122, 3128, 3135 |
| <code>\@@_Hline:</code> | 1078, 3874 |
| <code>\@@_Hline_i:n</code> | 3874, 3875, 3881 |
| <code>\@@_Hline_ii:nn</code> | 3878, 3881 |
| <code>\@@_Hline_iii:n</code> | 3879, 3882 |
| <code>\@@_Hspace:</code> | 1079, 3061 |
| <code>\@@_Iddots</code> | 1011, 1076, 3031 |
| <code>\g_@@_Iddots_lines_tl</code> | 1104, 2317 |
| <code>\@@_Ldots</code> | 1007, 1012, 1072, 2963 |
| <code>\g_@@_Ldots_lines_tl</code> | 1101, 2319 |
| <code>\l_@@_Matrix_bool</code> | 240, 1372, 1388, 2095 |
| <code>\l_@@_NiceArray_bool</code> | 237, 339, 1147, 1260, 1315, 1419, 1431, 2071, 3751, 3752, 3870, 3871, 4073, 4080, 4892 |
| <code>\g_@@_NiceMatrixBlock_int</code> | 228, 4134, 4139, 4142, 4153 |
| <code>\l_@@_NiceTabular_bool</code> | 150, 238, 778, 943, 1122, 1202, 1206, 1349, 1353, 1420, 1432, 2124, 2133, 4432, 4508, 4900 |
| <code>\@@_NotEmpty:</code> | 1089, 2118 |
| <code>\@@_OnlyMainNiceMatrix:n</code> | 1085, 3612 |
| <code>\@@_OnlyMainNiceMatrix_i:n</code> | 3615, 3622, 3625 |
| <code>\@@_Vdots</code> | 1009, 1074, 2993 |
| <code>\g_@@_Vdots_lines_tl</code> | 1102, 2315 |
| <code>\@@_Vdotsfor:</code> | 1081, 3211 |
| <code>\@@_Vdotsfor:nnnn</code> | 3215, 3226 |
| <code>\@@_W:</code> | 1392, 1461 |
| <code>\@@_actually_color:</code> | 1205, 3351 |
| <code>\@@_actually_diagbox:nnnnnn</code> | 4411, 4626, 4912, 4928 |
| <code>\@@_actually_draw_Cdots:</code> | 2565, 2569 |
| <code>\@@_actually_draw_Ddots:</code> | 2691, 2695 |
| <code>\@@_actually_draw_Iddots:</code> | 2742, 2746 |
| <code>\@@_actually_draw_Ldots:</code> | 2523, 2527, 3202 |
| <code>\@@_actually_draw_Vdots:</code> | 2618, 2622, 3277 |
| <code>\@@_adapt_S_column:</code> | 164, 179, 1121 |
| <code>\@@_add_to_colors_seq:nn</code> | 3338, 3350, 3372, 3387, 3402 |
| <code>\@@_adjust_pos_of_blocks_seq:</code> | 2231, 2281 |
| <code>\@@_adjust_pos_of_blocks_seq_i:nnnn</code> | 2284, 2286 |
| <code>\@@_adjust_size_box:</code> | 849, 875, 1552, 2007, 2051 |
| <code>\@@_after_array:</code> | 1381, 2137 |
| <code>\g_@@_after_col_zero_bool</code> | 263, 984, 1984, 3119 |
| <code>\@@_analyze_end:Nn</code> | 1782, 1827 |
| <code>\l_@@_argspec_tl</code> | 2961, 2962, 2963, 2978, 2993, 3008, 3031, 3133, 3134, 3135, 3209, 3210, 3211, 3287, 3288, 3289 |
| <code>\@@_array:</code> | 938, 1783, 1810 |
| <code>\l_@@_auto_columns_width_bool</code> | 444, 569, 1894, 1898, 4129 |
| <code>\l_@@_baseline_tl</code> | 433, 434, 562, 563, 564, 565, 951, 1317, 1595, 1607, 1612, 1614, 1619, 1624, 1706, 1707, 1711, 1716, 1718, 1723 |
| <code>\@@_begin_of_NiceMatrix:nn</code> | 2093, 2114 |
| <code>\@@_begin_of_row:</code> | 775, 799, 1985 |
| <code>\l_@@_block_auto_columns_width_bool</code> | 1135, 1899, 4122, 4127, 4137, 4147 |
| <code>\g_@@_block_box_int</code> | 290, 1115, 4406, 4420, 4422, 4452, 4464, 4476, 4485, 4495 |
| <code>\g_@@_blocks_dp_dim</code> | 234, 857, 860, 861, 4479, 4482 |
| <code>\g_@@_blocks_ht_dim</code> | 233, 863, 866, 867, 4470, 4473 |
| <code>\g_@@_blocks_seq</code> | 278, 1137, 1645, 4489, 4501, 4563 |
| <code>\g_@@_blocks_wd_dim</code> | 232, 851, 854, 855, 4458, 4461 |
| <code>\c_@@_booktabs_loaded_bool</code> | 24, 30, 1023, 1677 |
| <code>\@@_cartesian_path:</code> | 3381, 3396, 3417, 3502, 3514, 3549 |
| <code>\l_@@_cell_box</code> | 777, 823, 825, 831, 837, 840, 844, 853, 854, 859, 860, 865, 866, 876, 877, 878, 879, 881, 884, 888, 890, 908, 1025, |

| | |
|---|---|
| 1215, 1217, 1542, 1553, 1986, 2010, 2013, 2015, 2031, 2054, 2058, 4634, 4738, 4772, 4907 | \c_@@_define_L_C_R_bool ... 215, 1272, 4993 |
| \l_@@_cell_space_bottom_limit_dim 422, 490, 494, 879 | \@@_define_com:nnn 4876, 4884, 4885, 4886, 4887, 4888 |
| \l_@@_cell_space_top_limit_dim 421, 488, 495, 877 | \l_@@_delimiters_color_tl 455, 679, 725, 742, 1340, 1341 |
| \l_@@_cell_type_tl 230, 231, 1478, 1544, 4384, 4386 | \g_@@_delta_x_one_dim 2212, 2718, 2728 |
| \@@_cellcolor 1196, 3419, 3431, 3432 | \g_@@_delta_x_two_dim 2214, 2769, 2779 |
| \@@_cellcolor_tabular 1017, 3589 | \g_@@_delta_y_one_dim 2213, 2720, 2728 |
| \g_@@_cells_seq 1821, 1822, 1823, 1825 | \g_@@_delta_y_two_dim 2215, 2771, 2779 |
| \@@_chessboardcolors 1201, 3424 | \@@_diagbox:nn 1088, 4908 |
| \@@_cline 129, 1071 | \@@_dotfill: 4897 |
| \@@_cline_i:nn 130, 131, 143, 146 | \@@_dotfill_i: 4902, 4904 |
| \@@_cline_i:w 131, 132 | \@@_dotfill_ii: 4901, 4904, 4905 |
| \l_@@_code_before_bool 267, 559, 586, 958, 1143, 1153, 1841, 1858, 1876, 1907, 1933, 1960, 2183, 2275 | \@@_dotfill_iii: 4905, 4906 |
| \l_@@_code_before_tl .. 266, 558, 1144, 1204 | \@@_double_int_eval:n 3283, 3297, 3298 |
| \l_@@_code_for_first_col_tl 508, 1997 | \g_@@_dp_ante_last_row_dim 802, 1056 |
| \l_@@_code_for_first_row_tl . 512, 787, 4710 | \g_@@_dp_last_row_dim 802, 803, 1059, 1060, 1216, 1217, 1334 |
| \l_@@_code_for_last_col_tl 510, 2040 | \g_@@_dp_row_zero_dim 822, 823, 1050, 1051, 1327, 1700, 1739 |
| \l_@@_code_for_last_row_tl . 514, 794, 4713 | \@@_draw_Cdots:nnn 2551 |
| \g_@@_col_total_int 776, 1096, 1301, 1926, 1927, 1963, 1967, 1972, 1973, 2030, 2141, 2144, 2149, 2156, 2200, 2655, 3109, 3110, 3272, 4180, 4190, 4224, 4311, 4577, 4745 | \@@_draw_Ddots:nnn 2683 |
| \l_@@_color_tl 285, 4361, 4424, 4426 | \@@_draw_Iddots:nnn 2734 |
| \l_@@_colors_seq 1203, 3341, 3345, 3346, 3355 | \@@_draw_Ldots:nnn 2509 |
| \@@_colortbl_like: 1015, 1090 | \@@_draw_Vdots:nnn 2603 |
| \l_@@_colortbl_like_bool 419, 585, 1090, 1407 | \@@_draw_blocks: 1645, 4560 |
| \c_@@_colortbl_loaded_bool ... 82, 86, 1040 | \@@_draw_dotted_lines: 2229, 2303 |
| \l_@@_cols_tl 3380, 3395, 3416, 3441, 3449, 3450, 3551 | \@@_draw_dotted_lines_i: 2306, 2310 |
| \@@_columncolor 1200, 3383 | \l_@@_draw_first_bool . 289, 3023, 3046, 3057 |
| \@@_columncolor:n 3389, 3392 | \@@_draw_hlines: 2242, 3867 |
| \@@_columncolor_preamble 1019, 3603 | \@@_draw_line: 2549, 2601, 2681, 2732, 2783, 2785, 3336, 4090, 4120 |
| \c_@@_columncolor_regex 203, 1410 | \@@_draw_line_ii:nn 3316, 3320 |
| \l_@@_columns_width_dim 229, 570, 694, 1895, 1901, 4135, 4141 | \@@_draw_line_iii:nn 3323, 3327 |
| \g_@@_com_or_env_str 251, 254 | \@@_draw_non_standard_dotted_line: 2835, 2837 |
| \@@_computations_for_large_nodes: 4251, 4264, 4269 | \@@_draw_non_standard_dotted_line:n 2840, 2843 |
| \@@_computations_for_medium_nodes: 4171, 4240, 4250, 4261 | \@@_draw_standard_dotted_line: . 2834, 2856 |
| \@@_compute_a_corner:nnnnnn 3945, 3947, 3949, 3951, 3956 | \@@_draw_standard_dotted_line_i: 2875, 2879 |
| \@@_compute_corners: 2230, 3937 | \l_@@_draw_tl 284, 4543, 4548, 4599, 4780, 4786, 4788, 4790, 4821, 4822 |
| \@@_construct_preamble:n 1273, 1385 | \@@_draw_vlines: 2243, 3748 |
| \@@_create_col_nodes: 1786, 1814, 1833 | \g_@@_empty_cell_bool 274, 883, 892, 2020, 2066, 2976, 2991, 3006, 3029, 3052, 3063 |
| \@@_create_extra_nodes: 1644, 4161 | \l_@@_empty_corner_cells_seq 2236, 3686, 3692, 3699, 3807, 3813, 3820, 3939, 4012 |
| \@@_create_large_nodes: 4169, 4245 | \@@_end_Cell: 196, 870, 1480, 1532, 1548, 2104, 3078, 3079, 3080, 3081, 3082, 3083, 3084, 3085, 3086, 3087, 3088, 3089 |
| \@@_create_medium_and_large_nodes: 4166, 4256 | \l_@@_end_of_row_tl 452, 453, 502, 1806, 1807, 5157 |
| \@@_create_medium_nodes: 4167, 4235 | \c_@@_endpgfortikzpicture_tl 39, 43, 2307, 3324, 4059 |
| \@@_create_nodes: 4242, 4253, 4263, 4266, 4307 | \c_@@_enumitem_loaded_bool 25, 33, 312, 613, 618, 629, 634 |
| \@@_create_row_node: 954, 987, 1024 | \@@_env: 223, 227, 808, 814, 909, 915, 963, 969, 975, 1167, 1168, 1174, 1175, 1182, 1183, 1193, 1842, 1845, 1847, 1863, 1869, 1872, 1881, 1887, 1890, 1912, 1918, 1921, 1938, 1944, 1950, 1963, 1967, 1973, 2254, 2384, 2452, 2491, 2502, 3165, 3183, 3240, 3258, 3309, 3311, |
| \@@_cut_on_hyphen:w 3363, 3409, 3414, 3470, 3555, 3556, 3572, 3573, 4792, 4801 | |
| \g_@@_ddots_int 2210, 2715, 2716 | |
| \@@_def_env:nnn 2077, 2088, 2089, 2090, 2091, 2092 | |
| \@@_define_L_C_R: 216, 1272 | |

| | |
|--|---|
| 3330, 3333, 3967, 3986, 4004, 4193, 4195, | \@@_hdottedline_i:n 4058, 4062 |
| 4203, 4314, 4323, 4341, 4649, 4656, 4660, | \@@_hline:nn 3755, 3872, 3890 |
| 4674, 4679, 4691, 4696, 4697, 4700, 4717, 4751 | \@@_hline_i:nn 2240, 3758, 3761 |
| \g_@@_env_int .. 222, 223, 225, 1134, 1141, | \@@_hline_i_complete:nn 2240, 3865 |
| 1145, 1155, 1159, 1162, 1171, 1172, 1179, | \@@_hline_ii:nnnn ... 3783, 3794, 3827, 3866 |
| 1180, 1225, 1228, 1243, 1246, 2148, 2169, | \l_@@_hlines_bool 439, 516, 521, 551, 988, 2242 |
| 2187, 2190, 2203, 2271, 3457, 3460, 3485, 4350 | \g_@@_ht_last_row_dim |
| \@@_error:n 12, 315, 340, 465, | 804, 1057, 1058, 1214, 1215, 1333 |
| 475, 639, 682, 693, 702, 707, 727, 734, 744, | \g_@@_ht_row_one_dim .. 830, 831, 1054, 1055 |
| 750, 755, 766, 768, 1296, 1306, 1375, 1629, | \g_@@_ht_row_zero_dim |
| 1682, 1728, 3444, 4546, 4558, 5003, 5030, 5040 | 824, 825, 1052, 1053, 1328, 1699, 1738 |
| \@@_error:nn ... 13, 577, 1463, 1464, 1465, | \@@_i: 4173, 4175, |
| 2966, 2969, 2981, 2984, 2996, 2999, 3012, | 4176, 4177, 4178, 4187, 4193, 4195, 4196, |
| 3013, 3018, 3019, 3035, 3036, 3041, 3042, 3953 | 4197, 4198, 4203, 4204, 4205, 4206, 4214, |
| \@@_error:nnn 14, 3314 | 4217, 4219, 4220, 4221, 4273, 4275, 4278, |
| \@@_error_too_much_cols: 1442, 5053 | 4279, 4283, 4284, 4309, 4314, 4316, 4318, |
| \@@_everycr: 980, 1045, 1048 | 4322, 4323, 4334, 4341, 4343, 4345, 4349, 4350 |
| \@@_everycr_i: 980, 981 | \g_@@_iddots_int 2211, 2766, 2767 |
| \l_@@_except_corners_clist | \l_@@_in_env_bool 236, 339, 1125, 1126 |
| 440, 545, 549, 3652, 3774, 3940 | \c_@@_in_preamble_bool . 21, 22, 23, 609, 625 |
| \l_@@_exterior_arraycolsep_bool | \@@_info:n 5037 |
| 435, 690, 1422, 1434 | \l_@@_initial_i_int 2217, |
| \l_@@_extra_left_margin_dim | 2329, 2404, 2407, 2432, 2440, 2444, 2453, |
| 450, 537, 1276, 2018 | 2461, 2471, 2492, 2534, 2589, 2591, 2634, |
| \l_@@_extra_right_margin_dim | 2699, 2750, 3155, 3156, 3166, 3234, 3244, 3246 |
| 451, 538, 1288, 2062, 2658 | \l_@@_initial_j_int |
| \@@_extract_coords_values: 4332, 4339 | 2218, 2330, 2405, 2412, |
| \@@_fatal:n 15, 245, 1125, | 2417, 2423, 2433, 2441, 2445, 2454, 2462, |
| 1791, 1795, 1797, 1830, 3124, 5058, 5061, 5064 | 2472, 2493, 2531, 2573, 2648, 2650, 2655, |
| \@@_fatal:nn 16, 1471 | 2701, 2752, 3159, 3169, 3171, 3230, 3231, 3242 |
| \l_@@_fill_tl ... 283, 4541, 4547, 4612, 4617 | \l_@@_initial_open_bool |
| \l_@@_final_i_int | 2221, 2406, 2410, 2413, 2420, 2426, |
| 2219, 2331, 2336, 2339, 2364, | 2430, 2446, 2529, 2571, 2586, 2596, 2625, |
| 2372, 2376, 2385, 2393, 2473, 2503, 2543, | 2632, 2644, 2697, 2748, 2881, 2928, 3153, |
| 2640, 2707, 2758, 3156, 3184, 3252, 3262, 3264 | 3160, 3172, 3228, 3235, 3247, 3305, 4055, 4094 |
| \l_@@_final_j_int | \@@_insert_tabularnotes: 1649, 1652 |
| 2220, 2332, 2337, 2344, 2349, 2355, 2365, | \@@_instruction_of_type:nnn |
| 2373, 2377, 2386, 2394, 2474, 2504, 2540, | 919, 2971, 2986, 3001, 3023, 3046 |
| 2580, 2709, 2760, 3177, 3187, 3189, 3231, 3260 | \l_@@_inter_dots_dim |
| \l_@@_final_open_bool 2222, 2338, | . 423, 424, 2226, 2886, 2893, 2904, 2912, |
| 2342, 2345, 2352, 2358, 2362, 2378, 2538, | 2919, 2924, 2936, 2944, 4085, 4088, 4116, 4118 |
| 2578, 2587, 2598, 2625, 2638, 2646, 2667, | \g_@@_internal_code_after_tl 259, 1514, |
| 2705, 2756, 2883, 2898, 2929, 2930, 3154, | 1569, 2244, 2245, 3889, 4048, 4409, 4624, 4910 |
| 3178, 3190, 3229, 3253, 3265, 3306, 4056, 4095 | \@@_intersect_our_row:nnnn 3538 |
| \@@_find_extremities_of_line:nnnn ... | \@@_intersect_our_row_p:nnnn 3489 |
| 2326, 2513, 2555, 2607, 2687, 2738 | \@@_j: 4180, 4182, |
| \l_@@_first_col_int | 4183, 4184, 4185, 4190, 4193, 4195, 4198, |
| . 117, 130, 293, 294, 504, 748, 775, 1310, | 4200, 4201, 4203, 4206, 4208, 4209, 4224, |
| 1414, 1836, 1856, 2194, 3117, 3563, 3614, | 4227, 4229, 4230, 4231, 4286, 4288, 4291, |
| 4180, 4190, 4224, 4272, 4311, 4858, 4864, 4870 | 4293, 4297, 4298, 4311, 4314, 4315, 4317, |
| \l_@@_first_row_int | 4322, 4323, 4335, 4341, 4342, 4344, 4349, 4350 |
| 291, 292, 505, 752, 1094, | \l_@@_l_dim |
| 1325, 1626, 1697, 1725, 1736, 2192, 4173, | 2859, 2860, 2873, 2874, 2886, 2892, |
| 4187, 4214, 4271, 4309, 4653, 4671, 4856, 5175 | 2903, 2911, 2919, 2924, 2936, 2937, 2944, 2945 |
| \c_@@_footnote_bool | \l_@@_large_nodes_bool 447, 528, 4165, 4169 |
| 1111, 1383, 4975, 5001, 5024, 5043 | \g_@@_last_col_found_bool .. 301, 1099, |
| \c_@@_footnotehyper_bool . 4974, 5002, 5034 | 1302, 1367, 1925, 1954, 2028, 2140, 2197, 4743 |
| \@@_full_name_env: | \l_@@_last_col_int |
| 252, 5074, 5081, 5089, 5097, 5101, 5180, | 299, 300, 683, 718, 720, 735, 751, |
| 5201, 5220, 5225, 5230, 5237, 5256, 5265, 5401 | 767, 1165, 1239, 1245, 1252, 1305, 1426, |
| \@@_hdottedline: 1077, 4041 | 2100, 2102, 2141, 2144, 2196, 2612, 2653, |
| \@@_hdottedline:n 4049, 4053 | 2968, 2983, 3019, 3042, 4568, 4573, 4574, |
| \@@_hdottedline_i: 4044, 4046 | |

| | |
|---|---|
| 4577, 4580, 4607, 4619, 4630, 4645, 4674, 4679, 4687, 4702, 4860, 4866, 4872, 5057, 5075 | 1890, 1920, 1921, 1946, 1949, 1969, 1972, 2151, 2155, 2172, 2176, 4319, 4322, 4346, 4349 |
| \l_@@_last_col_without_value_bool 298, 717, 2142, 5060 | \g_@@_names_seq 235, 576, 578, 5512 |
| \l_@@_last_empty_column_int 3977, 3978, 3991, 3997, 4010 | \l_@@_nb_cols_int 4847, 4852, 4855, 4859, 4865, 4871 |
| \l_@@_last_empty_row_int 3959, 3960, 3973, 3994 | \l_@@_nb_rows_int 4846, 4851, 4862 |
| \l_@@_last_row_int 295, 296, 506, 792, 838, 992, 1163, 1210, 1220, 1227, 1234, 1290, 1294, 1297, 1309, 1331, 1808, 1809, 1993, 1994, 2037, 2038, 2163, 2518, 2560, 2998, 3013, 3036, 3620, 3628, 4567, 4570, 4571, 4576, 4607, 4619, 4629, 4643, 4701, 4712, 4868, 5255 | \@@_newcolumnntype 998, 1391, 1392 |
| \l_@@_last_row_without_value_bool 297, 1222, 1292, 2161 | \@@_node_for_multicolumn:nn 4330, 4337 |
| \l_@@_left_delim_dim 1258, 1262, 1267, 1774, 2016 | \@@_node_for_the_cell: 889, 895, 2014, 2063 |
| \l_@@_left_delim_tl 1113, 4084 | \@@_node_position: .. 1174, 1176, 1182, 1184 |
| \l_@@_left_margin_dim 448, 531, 1275, 2017, 4074, 4302 | \g_@@_not_empty_cell_bool 265, 887, 893, 2119 |
| \l_@@_letter_for_dotted_lines_str 701, 709, 710, 1469 | \@@_not_in_exterior:nnnn 3530 |
| \l_@@_light_syntax_bool 432, 500, 1278, 1283, 2164 | \@@_not_in_exterior_p:nnnn 3462 |
| \@@_light_syntax_i 1799, 1802 | \l_@@_notes_above_space_dim 441, 442 |
| \@@_line 2258, 3289 | \l_@@_notes_bottomrule_bool 597, 738, 761, 1675 |
| \@@_line_i:nn 3296, 3303 | \l_@@_notes_code_after_tl 595, 1684 |
| \l_@@_line_width_dim 286, 4550, 4600, 4781, 4813, 4824 | \l_@@_notes_code_before_tl 593, 1656 |
| \@@_line_with_light_syntax:n ... 1813, 1817 | \@@_notes_label_in_list:n 308, 327, 335, 605 |
| \@@_line_with_light_syntax_i:n 1812, 1818, 1819 | \@@_notes_label_in_tabular:n . 307, 348, 602 |
| \@@_math_toggle_token: 149, 872, 1987, 2004, 2032, 2048, 4952, 4956 | \l_@@_notes_para_bool .. 591, 736, 759, 1660 |
| \g_@@_max_cell_width_dim 880, 881, 1136, 1900, 4128, 4154 | \@@_notes_style:n 306, 309, 327, 335, 351, 356, 599 |
| \l_@@_max_delimiter_width_bool 456, 499, 1363 | \l_@@_nullify_dots_bool 443, 526, 2975, 2990, 3005, 3028, 3051 |
| \c_@@_max_l_dim 2873, 2878 | \l_@@_number_of_notes_int 305, 342, 352, 362 |
| \l_@@_medium_nodes_bool 446, 527, 4163, 4693 | \@@_old_CT@arc@ 1127, 2279 |
| \@@_message_hdotsfor: 5066, 5074, 5081, 5089 | \@@_old_arraycolsep_dim 275 |
| \@@_msg_new:nn 17, 5006, 5015, 5071, 5078, 5086, 5094, 5099, 5105, 5110, 5115, 5120, 5125, 5130, 5136, 5142, 5149, 5154, 5160, 5165, 5172, 5179, 5181, 5187, 5192, 5199, 5206, 5213, 5218, 5228, 5234, 5241, 5247, 5253, 5261, 5263, 5269, 5514 | \@@_old_cdots 1065, 2990 |
| \@@_msg_new:nnn 18, 4976, 5274, 5284, 5302, 5346, 5398, 5450, 5500 | \@@_old_ddots 1067, 3028 |
| \@@_msg_redirect_name:nn 19, 696, 1379, 4583, 4586 | \@@_old_dotfill 4896, 4899, 4907 |
| \@@_multicolumn:nnn 1082, 3067 | \@@_old_dotfill: 1086 |
| \g_@@_multicolumn_cells_seq 1092, 3097, 4198, 4206, 4328, 4658, 4676 | \l_@@_old_iRow_int 260, 1027, 2323 |
| \g_@@_multicolumn_sizes_seq 1093, 3099, 4329 | \@@_old_ialign: 953, 1061, 4562 |
| \g_@@_name_env_str 250, 255, 256, 1119, 1120, 1829, 2072, 2073, 2081, 2082, 2111, 2122, 2130, 2277, 4880, 5055 | \@@_old_iddots 1068, 3051 |
| \l_@@_name_str 445, 579, 810, 813, 911, 914, 971, 974, 1223, 1232, 1235, 1241, 1250, 1253, 1846, 1847, 1871, 1872, 1889, | \l_@@_old_jCol_int 261, 1030, 2324 |
| | \@@_old_ldots 1064, 2975 |
| | \@@_old_multicolumn 3066, 3073 |
| | \@@_old_pgful@check@rerun 75, 79 |
| | \@@_old_vdots 1066, 3005 |
| | \l_@@_parallelize_diags_bool 436, 437, 523, 2208, 2713, 2764 |
| | \@@_patch_preamble:n 1404, 1446, 1488, 1517, 1571, 1583 |
| | \@@_patch_preamble_i:n 1450, 1451, 1452, 1474 |
| | \@@_patch_preamble_ii:nn 1453, 1454, 1455, 1485 |
| | \@@_patch_preamble_iii:n . 1456, 1490, 1498 |
| | \@@_patch_preamble_iii_i:n 1493, 1495 |
| | \@@_patch_preamble_iv:nnn 1457, 1458, 1459, 1520 |
| | \@@_patch_preamble_ix:n 1576, 1586 |
| | \@@_patch_preamble_v:nnnn 1460, 1461, 1537 |
| | \@@_patch_preamble_vi:n 1462, 1559 |
| | \@@_patch_preamble_vii:n 1470, 1565 |
| | \@@_patch_preamble_viii:n 1483, 1535, 1557, 1563, 1573, 1589 |
| | \@@_pgf_rect_node:nnn 395, 4695 |
| | \@@_pgf_rect_node:nnnnn 370, 4313, 4340, 4648, 4690 |
| | \c_@@_pgfortikzpicture_tl 38, 42, 2305, 3322, 4057 |
| | \@@_picture_position: 1168, 1176, 1184 |

| | |
|---|---|
| <code>\l_@@_pos_of_block_tl</code> ... | 287, 288, 4355, 4357, 4359, 4385, 4386, 4388, 4431, 4435, 4442, 4492, 4515, 4517, 4528, 4531, 4552, 4554, 4556, 4719, 4731, 4742, 4746, 4753, 4765 |
| <code>\g_@@_pos_of_blocks_seq</code> | 279, 1138, 2204, 2234, 2283, 3100, 3646, 3768, 4024, 4595, 4920 |
| <code>\g_@@_pos_of_stroken_blocks_seq</code> | 281, 1139, 3650, 3772, 4609 |
| <code>\g_@@_pos_of_xdots_seq</code> | 280, 1140, 2235, 2469, 3648, 3770 |
| <code>@@_pre_array:</code> | 1021, 1257 |
| <code>\c_@@_preamble_first_col_tl</code> | 1415, 1980 |
| <code>\c_@@_preamble_last_col_tl</code> | 1427, 2024 |
| <code>\g_@@_preamble_tl</code> | 1389, 1399, 1402, 1412, 1415, 1424, 1427, 1436, 1441, 1476, 1487, 1500, 1522, 1539, 1561, 1567, 1580, 1588, 1783, 1810 |
| <code>@@_pred:n</code> | 118, 148, 2102, 3687, 3700, 3808, 3821 |
| <code>@@_provide_pgfsyspdfmark:</code> . | 204, 213, 1110 |
| <code>@@_put_box_in_flow:</code> | 1365, 1591, 1776 |
| <code>@@_put_box_in_flow_bis:nn</code> | 1364, 1743 |
| <code>@@_put_box_in_flow_i:</code> | 1597, 1599 |
| <code>@@_qpoint:n</code> | 226, 1602, 1604, 1616, 1632, 1691, 1693, 1709, 1720, 1731, 2531, 2534, 2540, 2543, 2573, 2580, 2589, 2591, 2634, 2640, 2648, 2650, 2699, 2701, 2707, 2709, 2750, 2752, 2758, 2760, 3330, 3333, 3562, 3566, 3579, 3581, 3710, 3712, 3714, 3831, 3833, 3835, 4065, 4069, 4076, 4112, 4115, 4117, 4219, 4229, 4639, 4641, 4643, 4645, 4667, 4687, 4715, 4797, 4799, 4806, 4808, 4933, 4935, 4938, 4940 |
| <code>\l_@@_radius_dim</code> | 427, 428, 1568, 2225, 2547, 2548, 2953, 4043, 4067, 4113, 4114 |
| <code>\l_@@_real_left_delim_dim</code> | 1745, 1760, 1775 |
| <code>\l_@@_real_right_delim_dim</code> | 1746, 1772, 1778 |
| <code>@@_rectanglecolor</code> | 1197, 3398, 3422 |
| <code>@@_rectanglecolor:nn</code> | 3404, 3407 |
| <code>@@_renew_NC@rewrite@S:</code> | 185, 187, 1098 |
| <code>@@_renew_dots:</code> | 1005, 1091 |
| <code>\l_@@_renew_dots_bool</code> | 524, 1091, 4995 |
| <code>@@_renew_matrix:</code> | 686, 4826, 4997 |
| <code>\l_@@_respect_blocks_bool</code> | 3439, 3454, 3482 |
| <code>@@_restore_iRow_jCol:</code> | 2278, 2321 |
| <code>@@_revtex_array:</code> | 930, 941 |
| <code>\c_@@_revtex_bool</code> | 46, 48, 51, 940 |
| <code>\l_@@_right_delim_dim</code> | 1259, 1263, 1269, 1777, 2060 |
| <code>\l_@@_right_delim_tl</code> | 1114, 4087 |
| <code>\l_@@_right_margin_dim</code> | 449, 533, 1287, 2061, 2657, 4081, 4305 |
| <code>@@_rotate:</code> | 1084, 3282 |
| <code>\g_@@_rotate_bool</code> | 241, 847, 874, 1551, 2006, 2050, 3282, 4431, 4449, 4454, 4514, 4527, 4635 |
| <code>@@_rotate_cell_box:</code> | 835, 874, 1551, 2006, 2050, 4635 |
| <code>\g_@@_row_of_col_done_bool</code> | 264, 985, 1118, 1855 |
| <code>\g_@@_row_total_int</code> | 1095, 1308, 1627, 1726, 2163, 2170, 2177, 2193, 3197, 4173, 4187, 4214, 4309, 4576, 4653, 4671, 5176 |
| <code>@@_rowcolor</code> | 1198, 3368 |
| <code>@@_rowcolor:n</code> | 3374, 3377 |
| <code>@@_rowcolor_tabular</code> | 1018, 3594 |
| <code>@@_rowcolors</code> | 1199, 3446 |
| <code>@@_rowcolors_i:nnnn</code> | 3490, 3525 |
| <code>\l_@@_rowcolors_restart_bool</code> ... | 3442, 3473 |
| <code>\g_@@_rows_seq</code> . | 1805, 1807, 1809, 1811, 1813 |
| <code>\l_@@_rows_tl</code> .. | 3379, 3394, 3415, 3492, 3568 |
| <code>\l_@@_rules_color_tl</code> .. | 262, 479, 1151, 1152 |
| <code>\@@_set_CT@arc@:</code> | 151, 1152 |
| <code>\@@_set_CT@arc@i:</code> | 152, 153 |
| <code>\@@_set_CT@arc@ii:</code> | 152, 155 |
| <code>@@_set_final_coords:</code> | 2482, 2507 |
| <code>@@_set_final_coords_from_anchor:n</code> . | 2498, 2546, 2584, 2628, 2643, 2712, 2763 |
| <code>@@_set_initial_coords:</code> | 2477, 2496 |
| <code>@@_set_initial_coords_from_anchor:n</code> . | 2487, 2537, 2577, 2627, 2637, 2704, 2755 |
| <code>@@_set_size:n</code> | 4844, 4853 |
| <code>\c_@@_siunitx_loaded_bool</code> | 157, 161, 166, 184 |
| <code>\l_@@_small_bool</code> | 684, 723, 731, 753, 781, 1033, 1988, 2033, 2223 |
| <code>@@_standard_cline</code> | 114, 1070 |
| <code>@@_standard_cline:w</code> | 114, 115 |
| <code>\l_@@_standard_cline_bool</code> .. | 420, 486, 1069 |
| <code>\c_@@_standard_tl</code> | 430, 431, 2833, 4089, 4119 |
| <code>\g_@@_static_num_of_col_int</code> | 282, 1374, 1405, 4580, 5090, 5102, 5221 |
| <code>\l_@@_stop_loop_bool</code> | 2333, 2334, 2366, 2379, 2388, 2401, 2402, 2434, 2447, 2456 |
| <code>@@_stroke_block:nnn</code> | 4604, 4777 |
| <code>@@_succ:n</code> | 143, 147, 963, 969, 1515, 1604, 1938, 1944, 1949, 1950, 1963, 1967, 1972, 1973, 2198, 2540, 2580, 2591, 2640, 2650, 2707, 2709, 2752, 2758, 3534, 3566, 3579, 3683, 3714, 3752, 3804, 3835, 3871, 3890, 4029, 4031, 4033, 4035, 4076, 4117, 4279, 4283, 4293, 4297, 4643, 4645, 4687, 4806, 4808, 4938, 4940 |
| <code>\l_@@_suffix_tl</code> | 4241, 4252, 4262, 4265, 4314, 4322, 4323, 4341, 4349, 4350 |
| <code>\c_@@_table_collect_begin_tl</code> . | 174, 176, 194 |
| <code>\c_@@_table_print_tl</code> | 177, 178, 196 |
| <code>\l_@@_tabular_width_dim</code> | 239, 946, 948, 1438, 2131 |
| <code>\l_@@_tabularnote_tl</code> | 304, 740, 763, 1648, 1657 |
| <code>\g_@@_tabularnotes_seq</code> | 303, 343, 1663, 1669, 1685 |
| <code>@@_test_if_cell_in_a_block:nn</code> | 3963, 3981, 3999, 4019 |
| <code>@@_test_if_cell_in_block:nnnnnnn</code> ... | 4025, 4027 |
| <code>@@_test_if_hline_in_block:nnnn</code> | 3769, 3771, 3893 |
| <code>@@_test_if_hline_in_stroken_block:nnnn</code> | 3773, 3915 |
| <code>@@_test_if_math_mode:</code> | 242, 1124, 2083 |
| <code>@@_test_if_vline_in_block:nnnn</code> | 3647, 3649, 3904 |
| <code>@@_test_if_vline_in_stroken_block:nnnn</code> | 3651, 3926 |
| <code>@@_test_in_corner_h:</code> | 3774, 3802 |
| <code>@@_test_in_corner_v:</code> | 3653, 3681 |

| | | | |
|--|--|--|--|
| <code>\l_@@_the_array_box</code> .. | 1271, 1274, 1642, 1643 | <code>\l</code> | 2092, 4887 |
| <code>\c_@@_tikz_loaded_bool</code> | 26, 37, 1188, 2246, 4096 | <code>\l</code> | 5069, 5074, 5081, 5089, 5090, 5101, 5102, 5144, 5175, 5176, 5180, 5189, 5220, 5221, 5222, 5230, 5236, 5249, 5256, 5257, 5265 |
| <code>\@@_true_c:</code> | 195, 1462 | A | |
| <code>\l_@@_type_of_col_tl</code> .. | 721, 722, 2112, 2114 | <code>\aboverulesep</code> | 1679 |
| <code>\c_@@_types_of_matrix_seq</code> | 5045, 5046, 5051, 5055 | <code>\addtocounter</code> | 360 |
| <code>\@@_update_for_first_and_last_row:</code> .. | 818, 882, 1212, 2008, 2052 | <code>\alph</code> | 306 |
| <code>\@@_use_arraybox_with_notes:</code> ... | 1322, 1704 | <code>\arraybackslash</code> | 1528 |
| <code>\@@_use_arraybox_with_notes_b:</code> . | 1319, 1688 | <code>\arraycolsep</code> | 532, 534, 536, 945, 1036, 1262, 1263, 1351, 1355, 4073, 4080 |
| <code>\@@_use_arraybox_with_notes_c:</code> | 1320, 1352, 1640, 1702, 1741 | <code>\arrayrulecolor</code> | 89 |
| <code>\@@_vdottedline:n</code> | 1570, 4092 | <code>\arrayrulewidth</code> | 122, 127, 139, 481, 809, 962, 964, 970, 993, 1346, 1358, 1400, 1508, 1581, 1696, 1735, 1862, 1864, 1870, 1880, 1882, 1888, 1911, 1913, 1919, 1937, 1939, 1945, 3564, 3565, 3567, 3580, 3582, 3722, 3723, 3725, 3736, 3742, 3844, 3855, 3861, 3886, 4154, 4781 |
| <code>\@@_vdottedline_i:n</code> | 4099, 4104, 4108 | <code>\arraystretch</code> | 1035, 4428, 4511, 4524 |
| <code>\@@_vline:nn</code> | 1515, 3630, 3753 | <code>\AtBeginDocument</code> | 23, 27, 67, 83, 158, 182, 310, 611, 627, 2301, 2959, 3131, 3207, 3285, 3318, 4051 |
| <code>\@@_vline_i:nn</code> | 2239, 3635, 3639 | <code>\AutoNiceMatrix</code> | 4889 |
| <code>\@@_vline_i_complete:nn</code> | 2239, 3746 | <code>\AutoNiceMatrixWithDelims</code> ... | 4849, 4881, 4893 |
| <code>\@@_vline_ii:nnnn</code> ... | 3662, 3673, 3706, 3747 | B | |
| <code>\l_@@_vlines_bool</code> | 438, 517, 520, 550, 1397, 1421, 1433, 1578, 2243 | <code>\baselineskip</code> | 92, 99 |
| <code>\@@_w:</code> | 1391, 1460 | <code>\bgroup</code> | 1112 |
| <code>\g_@@_width_first_col_dim</code> | 277, 1117, 1313, 1850, 2009, 2010 | <code>\Block</code> | 1083, 5144, 5277 |
| <code>\g_@@_width_last_col_dim</code> | 276, 1116, 1369, 1959, 2053, 2054 | <code>\BNiceMatrix</code> | 4841 |
| <code>\l_@@_x_final_dim</code> | 270, 2484, 2541, 2542, 2581, 2582, 2630, 2652, 2660, 2664, 2668, 2670, 2675, 2677, 2710, 2719, 2727, 2761, 2770, 2778, 2796, 2805, 2853, 2866, 2918, 2934, 3334, 4077, 4088, 4114 | <code>\bNiceMatrix</code> | 4838 |
| <code>\l_@@_x_initial_dim</code> | 268, 2479, 2532, 2533, 2574, 2575, 2630, 2651, 2652, 2659, 2664, 2668, 2670, 2672, 2675, 2677, 2702, 2719, 2727, 2753, 2770, 2778, 2795, 2805, 2851, 2866, 2918, 2932, 2934, 2952, 2954, 3331, 4070, 4085, 4113 | bool commands: | |
| <code>\l_@@_xdots_color_tl</code> | 454, 468, 2522, 2564, 2616, 2617, 2690, 2741, 2841, 3201, 3276, 3293 | <code>\bool_do_until:Nn</code> | 2334, 2402 |
| <code>\l_@@_xdots_down_tl</code> | 472, 2789, 2824 | <code>\bool_gset_false:N</code> | 847, 892, 893, 984, 1099, 1118, 2020, 2066, 3902, 3913, 3924, 3935, 4454 |
| <code>\l_@@_xdots_line_style_tl</code> | 429, 431, 464, 2833, 2841, 4089, 4119 | <code>\bool_gset_true:N</code> | 1855, 1984, 2028, 2119, 2976, 2991, 3006, 3029, 3052, 3063, 3282, 3645, 3767 |
| <code>\l_@@_xdots_shorten_dim</code> | 425, 426, 470, 2227, 2848, 2849, 2892, 2903, 2911 | <code>\bool_if:NTF</code> | 150, 166, 613, 618, 629, 634, 778, 781, 874, 958, 985, 988, 1023, 1033, 1090, 1091, 1111, 1125, 1135, 1153, 1188, 1202, 1206, 1272, 1292, 1367, 1372, 1383, 1407, 1551, 1578, 1675, 1841, 1858, 1876, 1894, 1907, 1933, 1954, 1960, 1988, 2006, 2033, 2050, 2140, 2142, 2161, 2164, 2183, 2208, 2223, 2242, 2243, 2246, 2596, 2598, 2713, 2764, 2975, 2990, 3005, 3028, 3051, 3972, 3990, 4008, 4137, 4147, 4169, 4431, 4449, 4514, 4527, 4635, 4693, 4900, 5024, 5034, 5060 |
| <code>\l_@@_xdots_up_tl</code> | 473, 2788, 2814 | <code>\bool_if:nTF</code> | 184, 312, 339, 921, 1302, 3307, 3540, 4163, 4743 |
| <code>\l_@@_y_final_dim</code> | 271, 2485, 2544, 2548, 2593, 2597, 2599, 2641, 2708, 2721, 2724, 2759, 2772, 2775, 2796, 2804, 2853, 2868, 2923, 2942, 3335, 4068, 4118 | <code>\bool_lazy_all:nTF</code> | 1417, 1429, 2232, 3895, 3906, 3917, 3928 |
| <code>\l_@@_y_initial_dim</code> | 269, 2480, 2535, 2547, 2592, 2593, 2597, 2599, 2635, 2700, 2721, 2726, 2751, 2772, 2777, 2795, 2804, 2851, 2868, 2923, 2940, 2942, 2952, 2955, 3332, 4066, 4067, 4068, 4116 | <code>\bool_lazy_and:nnTF</code> ... | 1897, 1989, 2195, 2585, 2787, 3115, 3453, 3481, 3716, 3837, 4793 |
| <code>\l</code> | 1796, 1818, 4860, 4866, 4872, 4978, 4979, 5012, 5021, 5102, 5107, 5112, 5117, 5122, 5145, 5156, 5162, 5169, 5176, 5184, 5196, 5202, 5209, 5216, 5225, 5231, 5238, 5244, 5250, 5262, 5266, 5271, 5277, 5286, 5287, 5305, 5306, 5349, 5350, 5401, 5402, 5453, 5454, 5507, 5508 | <code>\bool_lazy_or:nnTF</code> | 461, 886, 1625, 1646, 1724, 2036, 2625, 2872, 3532, 3964, 3982, 4000, 4372, 4377, 4397, 4575, 4598 |
| <code>\{</code> | 256, 2090, 4888, 5183, 5238, 5349, 5453 | | |
| <code>\}</code> | 256, 2090, 4888, 5183, 5238, 5349, 5453 | | |

`\bool_lazy_or_p:nn` 1992
`\bool_not_p:n`
1420, 1421, 1422, 1432, 1433, 1434, 1899, 2197
`\bool_set:Nn` 2629, 3475
`\c_false_bool` 2971, 2986, 3001
`\g_tmpa_bool`
.... 3645, 3654, 3688, 3696, 3701, 3767,
3775, 3809, 3817, 3822, 3902, 3913, 3924, 3935
`\l_tmpb_bool` ... 3969, 3983, 4001, 4023, 4036
box commands:
`\box_clear_new:N` 1025, 1271
`\box_dp:N` 803, 823, 860,
879, 1051, 1060, 1217, 1594, 1754, 1767, 4484
`\box_gclear_new:N` 4419
`\box_grotate:Nn` 4451
`\box_ht:N` ... 804, 825, 831, 843, 866, 877,
1053, 1055, 1058, 1215, 1593, 1754, 1767, 4475
`\box_move_up:nn` . 58, 60, 62, 1637, 1702, 1741
`\box_rotate:Nn` 837
`\box_set_dp:Nn` 859, 878, 1594
`\box_set_ht:Nn` 865, 876, 1593
`\box_set_wd:Nn` 853
`\box_use:N` 363, 844, 1529, 1532
`\box_use_drop:N` 884, 890, 908, 1553,
1596, 1637, 1638, 1643, 2015, 4494, 4738, 4772
`\box_wd:N`
... 364, 854, 881, 888, 1267, 1269, 1642,
1761, 1773, 2010, 2013, 2054, 2058, 4463, 4907
`\l_tmpa_box`
.. 346, 363, 364, 1266, 1267, 1268, 1269,
1337, 1593, 1594, 1596, 1637, 1638, 1754, 1767
`\l_tmpb_box` 1747, 1761, 1762, 1773

C

`\c` 203, 1411
`\Cdots` 1073, 2981, 2984
`\cdots` 1008, 1065
`\cellcolor` 1017, 1196, 3592
`\chessboardcolors` 1201
`\cline` 142, 1070, 1071
clist commands:
`\clist_if_empty:NTF` 3652, 3774
`\clist_map_inline:Nn` 3551, 3568, 3940
`\clist_map_inline:nn` 2107, 3421, 3466
`\clist_new:N` 440
`\clist_set:Nn` 549
`\CodeAfter` 772, 1087, 1799, 1802, 2259
`\color` 93, 100, 154, 156, 1341, 2516, 2519, 2522,
2558, 2561, 2564, 2610, 2613, 2617, 2690,
2741, 3195, 3198, 3201, 3270, 3273, 3276,
3293, 3357, 3499, 3500, 3511, 3512, 4426, 4548
`\colorlet` 248, 249, 788, 795, 1998, 2041
`\columncolor` 1019, 1200, 2266, 3608
`\cr` 126, 144, 1978
`\crr` 1835
cs commands:
`\cs_generate_variant:Nn` 146, 3350, 4159, 4160
`\cs_gset:Npn`
.... 93, 100, 2148, 2155, 2169, 2176, 4152
`\cs_gset_eq:NN` ... 179, 213, 1044, 1127, 2279
`\cs_if_exist:NTF`
1026, 1029, 1128, 1131, 1225, 1232, 1243,
1250, 2323, 2324, 2369, 2382, 2437, 2450,
3163, 3181, 3238, 3256, 4139, 4192, 4655, 4673

`\cs_if_exist_p:N`
.... 462, 3456, 3484, 3966, 3985, 4003
`\cs_if_free:NTF`
.... 209, 2511, 2553, 2605, 2685, 2736
`\cs_if_free_p:N` 3309, 3311
`\cs_new_protected:Npx` 2303, 3320, 4053
`\cs_set:Nn` 599, 602, 605
`\cs_set:Npn` 89, 90,
96, 97, 102, 114, 115, 129, 131, 132, 154,
156, 309, 1000, 2328, 2390, 2458, 3205,
3280, 3874, 3875, 3881, 3882, 4428, 4511, 4524
`\cs_set_nopar:Npn`
.... 935, 947, 1035, 1038, 4334, 4335
`\cs_set_nopar:Npx` 948
`\cs_set_protected:Npn` 4878
`\cs_set_protected_nopar:Npn` 4407, 4622

D

`\Ddots` 1075, 3012, 3013, 3018, 3019
`\ddots` 1010, 1067
`\diagbox` 1088, 4407, 4622
dim commands:
`\dim_add:Nn` 4303
`\dim_compare:nNnTF`
.... 92, 99, 851, 857, 863, 1438,
1895, 2058, 2670, 4216, 4226, 4665, 4685, 4907
`\dim_gzero:N` 855, 861, 867
`\dim_max:nn` 4205, 4209
`\dim_min:nn` 4197, 4201
`\dim_ratio:nn` 2728, 2779,
2886, 2891, 2902, 2910, 2919, 2924, 2935, 2943
`\dim_set:Nn` 4178, 4185, 4196, 4200,
4204, 4208, 4220, 4221, 4230, 4231, 4275, 4288
`\dim_set_eq:NN` 4176, 4183, 4283, 4297
`\dim_sub:Nn` 4300
`\dim_use:N`
.... 4217, 4227, 4278, 4279, 4291, 4292,
4315, 4316, 4317, 4318, 4342, 4343, 4344, 4345
`\dim_zero_new:N` 4175, 4177, 4182, 4184
`\c_max_dim` 4176, 4178,
4183, 4185, 4217, 4227, 4652, 4665, 4670, 4685
`\l_tmpc_dim`
.... 272, 3564, 3565, 3584, 3715, 3723,
3728, 3733, 3739, 3836, 3847, 3852, 3858,
4644, 4650, 4692, 4800, 4811, 4939, 4942, 4950
`\l_tmpd_dim` 273, 3582, 3584, 3724,
3728, 3843, 3847, 4646, 4650, 4670, 4681,
4685, 4688, 4692, 4809, 4812, 4941, 4942, 4954
`\dotfill` 1086, 4896
`\dots` 1012
`\doublerulesep` 1509, 3725, 3737, 3844, 3856, 3887
`\doublerulesepcolor` 96
`\draw` 2845

E

`\egroup` 1382
else commands:
`\else:` 244
`\endarray` 1787, 1815
`\endBNiceMatrix` 4842
`\endbNiceMatrix` 4839
`\endNiceArray` 2127, 2136
`\endNiceArrayWithDelims` 2076, 2086
`\endpgfscope` 2829, 4953

\leaders 122, 139
\left 1342, 1750, 1765
legacy commands:
\legacy_if:nTF 573
\line 2258, 5183

M

\makebox 1553
\mathinner 55
mode commands:
\mode_leave_vertical: 1123, 1527
msg commands:
\msg_error:nn 12
\msg_error:nnn 13
\msg_error:nnnn 14, 4582, 4589
\msg_fatal:nn 15
\msg_fatal:nnn 16
\msg_info:nn 5027
\msg_new:nnn 17
\msg_new:nnnn 18
\msg_redirect_name:nnn 20
\multicolumn . 1082, 3066, 3070, 3121, 3127, 3148
\multispan 118, 119, 135, 136
\myfiledate 6
\myfileversion 7

N

\newcolumntype 218, 219, 220
\newcounter 302
\NewDocumentCommand
. . 314, 337, 711, 2963, 2978, 2993, 3008,
3031, 3135, 3211, 3289, 3368, 3383, 3398,
3419, 3424, 3446, 3589, 3594, 3603, 4849, 4889
\NewDocumentEnvironment 1108,
1780, 1789, 2069, 2079, 2109, 2120, 2128, 4132
\NewExpandableDocumentCommand 224, 4364
\newlist 318, 329
\NiceArray 2125, 2134
\NiceArrayWithDelims 2074, 2084
nicematrix commands:
\g_nicematrix_code_after_tl
258, 582, 1804, 2260, 2261, 4602, 4962, 4970
\g_nicematrix_code_before_tl
. . . 1106, 2263, 2272, 3591, 3596, 3607, 4614
\NiceMatrixLastEnv 224
\NiceMatrixOptions 711, 5305, 5507
\NiceMatrixoptions 5208
\noalign .. 92, 99, 104, 127, 980, 1044, 3874, 4043
\nobreak 332
\normalbaselines 1032
\NotEmpty 1089
\nulldelimiterspace 1761, 1773
\numexpr 147, 148

O

\omit 117, 1838, 1854, 1930, 4959
\OnlyMainNiceMatrix 1085, 3611

P

\par 1657, 1665
peek commands:
\peek_meaning:NTF 152, 344, 1001
\peek_meaning_ignore_spaces:NTF 1782, 3877
\peek_meaning_remove_ignore_spaces:NTF 142

\peek_remove_spaces:n 3093
\pgfextracty 4715
\pgfgetlastxy 406
\pgfpathcircle 2951
\pgfpathlineto 3733, 3739, 3852, 3858, 4942
\pgfpathmoveto 3732, 3738, 3851, 3857, 4937
\pgfpathrectanglecorners 3583, 3726, 3845, 4810
\pgfpointadd 404
\pgfpointanchor 227, 2489, 2500,
4195, 4203, 4660, 4678, 4697, 4699, 4716, 4750
\pgfpointdiff 405, 1176, 1184
\pgfpointlineattime 2794
\pgfpointorigin 1845, 1968
\pgfpointscale 404
\pgfpointshapeborder 3330, 3333
\pgfrememberpicturepositiononpagetrue . .

. . 806, 899, 968, 1844, 1868, 1886, 1917,
1943, 1966, 2312, 2831, 3329, 3708, 3829,
4064, 4111, 4238, 4248, 4259, 4637, 4784, 4932
\pgfscope 2791, 4949
\pgfset 373, 398, 900, 4727, 4761, 4948
\pgfsetbaseline 898
\pgfsetlinewidth 3742, 3861, 4813
\pgfsetrectcap 3743, 3862
\pgfsetroundcap 4945
\pgfsetstrokecolor 4790
\pgfsyspdfmark 209, 210
\pgftransformrotate 2798
\pgftransformshift
. . . 379, 404, 2792, 4726, 4748, 4950, 4954
\pgfusepath 2818, 2828
\pgfusepathqfill
. . . . 2957, 3359, 3503, 3515, 3729, 3848
\pgfusepathqstroke 3744, 3863, 4814, 4946
\phantom 2975, 2990, 3005, 3028, 3051
\pNiceMatrix 4829

prg commands:

\prg_do_nothing:
. . . . 170, 179, 185, 213, 474, 1044, 2259
\prg_new_conditional:Nnn 3530, 3538
\prg_replicate:nn 352, 1926,
1927, 3148, 3734, 3853, 4859, 4862, 4865, 4871
\prg_return_false: 3535, 3547
\prg_return_true: 3536, 3546
\ProcessKeysOptions 5005
\ProvideDocumentCommand 53
\ProvidesExplPackage 4

Q

\quad 333

quark commands:

\q_stop
114, 115, 131, 132, 153, 155, 1152, 1404,
1466, 1799, 1802, 3283, 3297, 3298, 3363,
3409, 3414, 3470, 3555, 3556, 3572, 3573,
4332, 4339, 4366, 4369, 4792, 4801, 4844, 4853

R

\rectanglecolor 1197, 2265, 3598, 4616
\refstepcounter 361

regex commands:

\regex_const:Nn 203
\regex_replace_all:NnN 1409
\relax 147, 148

`\renewcommand` 189
`\RenewDocumentEnvironment`
 4828, 4831, 4834, 4837, 4840
`\RequirePackage` 1, 3, 9, 10, 11
`\right` 1360, 1757, 1769
`\rotate` 1084
`\rowcolor` 1018, 1198
`\rowcolors` 1199

S

`\savenotes` 1111
`\scriptstyle` 781, 1988, 2033, 2814, 2824
 seq commands:
`\seq_clear_new:N` 1203, 2187, 3939
`\seq_count:N` 1809, 3346
`\seq_gclear:N` .. 1137, 1138, 1139, 1140, 1685
`\seq_gclear_new:N` ... 1092, 1093, 1805, 1821
`\seq_gpop_left:NN` 1811, 1823
`\seq_gput_left:Nn` 578, 3097, 3099, 4595
`\seq_gput_right:Nn`
 343, 2469, 3100, 4489, 4501, 4609, 4920
`\seq_gset_from_clist:Nn` 2190, 2202
`\seq_gset_map_x:NNn` 2283
`\seq_gset_split:Nnn` 1807, 1822
`\seq_if_empty:NTF` 1645
`\seq_if_empty_p:N` 2234, 2235, 2236
`\seq_if_exist:NTF` 1155
`\seq_if_in:NnTF` .. 576, 3685, 3691, 3698,
 3806, 3812, 3819, 4198, 4206, 4658, 4676, 5055
`\seq_item:Nn` 1159, 1162, 1171, 1172, 1179, 1180
`\seq_map_function:NN` 1813
`\seq_map_indexed_inline:Nn` 3341, 3355
`\seq_map_inline:Nn` 1663, 1669, 1825, 3490,
 3646, 3648, 3650, 3768, 3770, 3772, 4024, 4563
`\seq_mapthread_function:NNN` 4327
`\seq_new:N` 235, 278, 279, 280, 281, 303, 5045
`\seq_put_right:Nn` 3345, 4011
`\seq_set_eq:NN` 3459
`\seq_set_filter:NNn` 3461, 3488
`\seq_set_from_clist:Nn` 5046
`\seq_set_map_x:NNn` 5051
`\seq_use:Nnnn` 2204, 5512
`\l_tmpa_seq` 3461, 3488
`\l_tmpb_seq` 3459, 3461, 3488, 3490
`\setlist` 319, 330, 614, 619, 630, 635
 skip commands:
`\skip_gadd:Nn` 1902
`\skip_gset:Nn` 1893
`\skip_gset_eq:NN` 1900, 1901
`\skip_horizontal:N`
 . 123, 140, 1275, 1276, 1287, 1288, 1312,
 1313, 1350, 1351, 1354, 1355, 1369, 1370,
 1400, 1568, 1581, 1774, 1775, 1777, 1778,
 1849, 1850, 1862, 1864, 1880, 1882, 1904,
 1911, 1913, 1932, 1937, 1939, 1958, 1959,
 2016, 2017, 2018, 2021, 2055, 2060, 2061, 2062
`\skip_horizontal:n` 364, 1504
`\skip_vertical:N` 127, 962,
 964, 1345, 1346, 1357, 1358, 1654, 1679, 4043
`\skip_vertical:n` 843, 3884
`\g_tmpa_skip` 1893, 1900, 1901, 1902, 1904, 1932
`\c_zero_skip` 1049
`\space` 255, 256
`\stepcounter` 350, 355

str commands:
`\c_backslash_str` 255
`\c_colon_str` 710
`\str_case:nn` ... 3076, 4719, 4731, 4753, 4765
`\str_case:nnTF` 1317, 1448, 1619, 3942
`\str_foldcase:n` 3076
`\str_gclear:N` 2277
`\str_gset:Nn`
 ... 1120, 2073, 2082, 2111, 2122, 2130, 4880
`\str_if_empty:NTF` 810, 911, 971,
 1119, 1223, 1241, 1846, 1871, 1889, 1920,
 1946, 1969, 2072, 2081, 2151, 2172, 4319, 4346
`\str_if_eq:nnTF` 78, 254, 951,
 1469, 1497, 1575, 1794, 1796, 1829, 4788, 4967
`\str_if_eq_p:nn` 463, 4374, 4379
`\str_if_in:NnTF` 1607, 1711
`\str_new:N` 250, 445, 709
`\str_range:Nnn` 1611, 1715
`\str_set:Nn` 575, 701
`\str_set_eq:NN` 579, 710
`\l_tmpa_str` 575, 576, 578, 579
`\strut` 1663, 1669

T

`\tabcolsep` 944, 1350, 1354
`\tabskip` 1049
`\tabularnote` 314, 337, 344, 5236, 5249
`\tabularnotes` 1668
 T_EX and L^AT_EX 2_ε commands:
`\@BTnormal` 1024
`\@acol` 934
`\@acoll` 932
`\@acolr` 933
`\@array@array` 936
`\@arrayacol` 932, 933, 934
`\@arstrutbox` 803, 804,
 843, 1051, 1053, 1055, 1058, 1060, 1529, 1532
`\@currenvir` 4967
`\@gobblethree` 210
`\@halignto` 935, 947, 948
`\@height` 107, 122, 139
`\@ifclassloaded` 47, 50, 5026, 5036
`\@ifnextchar` 1097
`\@ifpackageloaded`
 29, 32, 35, 69, 85, 160, 5029, 5039
`\@mainaux` 70, 206, 2145, 2146, 2153,
 2159, 2166, 2167, 2174, 2180, 2185, 2186,
 2188, 2206, 2267, 2268, 2274, 4149, 4150, 4156
`\@tabarray` 949
`\@tempswafalse` 1395
`\@tempwattrue` 1394
`\@temptokena` .. 169, 172, 191, 193, 1393, 1404
`\@whiles` 1395
`\@width` 107
`\@xhline` 110
`\bBigg@` 1266, 1268
`\c@MaxMatrixCols` 2101, 5083
`\c@tabularnote` 1647, 1658, 1686
`\col@sep` 944, 945, 1312, 1370, 1849,
 1902, 1958, 2021, 2055, 2533, 2542, 2575, 2582
`\CT@arc` 89, 90
`\CT@arc@` .. 88, 93, 108, 121, 138, 154, 156,
 1127, 1680, 2279, 3741, 3860, 4110, 4789, 4944
`\CT@dr` 96, 97

| | |
|---|---|
| <code>\CT@drsc@</code> . . . | 95, 100, 3718, 3721, 3839, 3842 |
| <code>\CT@everycr</code> | 1042 |
| <code>\CT@row@color</code> | 1044 |
| <code>\if@tempswa</code> | 1395 |
| <code>\NC@</code> | 1000 |
| <code>\NC@find</code> | 170, 198 |
| <code>\NC@list</code> | 1395 |
| <code>\NC@rewrite@S</code> | 171, 189 |
| <code>\new@ifnextchar</code> | 1097 |
| <code>\newcol@</code> | 1002, 1003 |
| <code>\nicematrix@redefine@check@rerun</code> . . | 70, 73 |
| <code>\pgf@relevantforpicturesizefalse</code> . . . | 2313, 2832, 2948, 3354, 3465, 3709, 3830, 4239, 4249, 4260, 4638, 4785, 4931 |
| <code>\pgfsys@getposition</code> | 1168, 1174, 1182 |
| <code>\pgfsys@markposition</code> | 963, 1167, 1842, 1863, 1881, 1912, 1938, 1962 |
| <code>\pgfutil@check@rerun</code> | 75, 76 |
| <code>\reserved@a</code> | 109 |
| <code>\set@color</code> | 4425 |
| <code>\tikz@library@external@loaded</code> | 1128 |
| tex commands: | |
| <code>\tex_mkern:D</code> | 57, 59, 61, 64 |
| <code>\tex_the:D</code> | 193 |
| <code>\textfont</code> | 1633 |
| <code>\textit</code> | 306 |
| <code>\textsuperscript</code> | 307, 308 |
| <code>\the</code> | 147, 148, 1395, 1404 |
| <code>\thetabularnote</code> | 309 |
| <code>\tikzexternaldisable</code> | 1130 |
| <code>\tikzset</code> | 1132, 1190, 2248 |
| tl commands: | |
| <code>\tl_clear:N</code> | 3667, 3678, 3788, 3799, 4780 |
| <code>\tl_clear_new:N</code> | 3410, 3411, 3449, 3642, 3764 |
| <code>\tl_const:Nn</code> | 38, 39, 42, 43, 430, 1980, 2024 |
| <code>\tl_count:n</code> | 1614, 1718 |
| <code>\tl_gclear:N</code> | 1402, 2245, 2261 |
| <code>\tl_gclear_new:N</code> | 1100, 1101, 1102, 1103, 1104, 1105, 1106 |
| <code>\tl_gput_left:Nn</code> | 921, 1415, 1424, 3607 |
| <code>\tl_gput_right:Nn</code> | 921, 1427, 1436, 1440, 1476, 1487, 1500, 1514, 1522, 1539, 1561, 1567, 1569, 1580, 1588, 1804, 3137, 3213, 3591, 3596, 3889, 4048, 4409, 4602, 4614, 4624, 4910, 4962, 4970 |
| <code>\tl_gset:Nn</code> | 172, 176, 178, 1389, 1399, 2270 |
| <code>\tl_if_blank:nTF</code> | 3370, 3373, 3385, 3388, 3400, 3403, 3496, 3508, 4366 |
| <code>\tl_if_blank_p:n</code> | 3718, 3839, 4373, 4378 |
| <code>\tl_if_empty:NnTF</code> | 1151, 1340, 1657, 2263, 3557, 3558, 3574, 3575, 3656, 3660, 3671, 3777, 3781, 3792, 4384, 4424, 4612, 4786 |
| <code>\tl_if_empty:nTF</code> | 556, 681, 715, 733, 749, 765, 1791, 1818, 2522, 2564, 2616, 2690, 2741, 3201, 3276, 3293, 3498, 3510, 5068 |
| <code>\tl_if_empty_p:N</code> | 2788, 2789, 4599 |
| <code>\tl_if_empty_p:n</code> | 1648 |
| <code>\tl_if_eq:NNTF</code> | 2833, 4084, 4087 |
| <code>\tl_if_eq:NnTF</code> | 1595, 1706 |
| <code>\tl_if_eq:nnTF</code> | 568, 692, 3342 |
| <code>\tl_if_exist:NnTF</code> | 1141 |
| <code>\tl_if_in:NnTF</code> | 3469, 3554, 3571 |
| <code>\tl_if_single_token:nTF</code> | 700 |
| <code>\tl_item:Nn</code> | 175, 176, 178 |
| <code>\tl_map_inline:nn</code> | 1792 |
| <code>\tl_new:N</code> | 174, 177, 230, 258, 259, 262, 266, 283, 284, 285, 287, 304, 429, 433, 452, 454, 455 |
| <code>\tl_put_left:Nn</code> | 1024 |
| <code>\tl_put_right:Nn</code> | 558, 1144, 1212, 3348 |
| <code>\tl_range:nnn</code> | 78 |
| <code>\tl_set:Nn</code> | 175, 3346, 3415, 3416, 3471, 3492, 3559, 3561, 3576, 3578, 3641, 4390, 4803, 4805 |
| <code>\tl_set_eq:NN</code> | 431, 3412, 3413, 3657, 3778, 4089, 4119, 4386 |
| <code>\tl_set_rescan:Nnn</code> | 1806, 2962, 3134, 3210, 3288 |
| <code>\tl_to_str:n</code> | 5052 |
| <code>\g_tmpa_tl</code> | 172, 175, 178 |
| <code>\l_tmpb_tl</code> | 3366, 3413, 3416, 3471, 3477, 3558, 3559, 3560, 3561, 3566, 3575, 3576, 3577, 3578, 3579, 3641, 3683, 3687, 3693, 3695, 3700, 3765, 3778, 3787, 3808, 3814, 3821, 3899, 3900, 3910, 3911, 3921, 3922, 3932, 3933, 4795, 4799, 4804, 4805, 4808 |
| <code>\l_tmpc_tl</code> | 3410, 3412, 3415, 3642, 3656, 3657, 3660, 3665, 3667, 3671, 3676, 3678, 3764, 3777, 3778, 3781, 3786, 3788, 3792, 3797, 3799 |
| <code>\l_tmpd_tl</code> | 3411, 3413, 3416 |
| token commands: | |
| <code>\token_to_str:N</code> | 5069, 5144, 5151, 5156, 5183, 5189, 5236, 5249, 5266, 5276, 5305, 5507 |
| U | |
| <code>\unskip</code> | 1673 |
| use commands: | |
| <code>\use:N</code> | 924, 1228, 1235, 1246, 1253, 1279, 1280, 1284, 1285, 2096, 2116, 3358 |
| <code>\use:n</code> | 3294, 3611 |
| <code>\usepackage</code> | 5031, 5041 |
| <code>\usepgfmodule</code> | 2 |
| V | |
| vbox commands: | |
| <code>\vbox:n</code> | 63 |
| <code>\vbox_set_top:Nn</code> | 840 |
| <code>\vbox_to_ht:nn</code> | 384, 411, 1753, 1766 |
| <code>\vbox_to_zero:n</code> | 842 |
| <code>\vcenter</code> | 1343, 1751, 5266 |
| <code>\Vdots</code> | 1074, 2996, 2999 |
| <code>\vdots</code> | 1009, 1066 |
| <code>\Vdotsfor</code> | 1081 |
| <code>\vfill</code> | 387, 413 |
| <code>\vline</code> | 108 |
| <code>\VNiceMatrix</code> | 4835 |
| <code>\vNiceMatrix</code> | 4832 |
| <code>\vrule</code> | 106 |
| <code>\vskip</code> | 105 |
| <code>\vtop</code> | 960 |
| X | |
| <code>\xglobal</code> | 788, 795, 1998, 2041 |

Contents

| | | |
|-----------|--|-----------|
| 1 | The environments of this package | 2 |
| 2 | The vertical space between the rows | 2 |
| 3 | The vertical position of the arrays | 3 |
| 4 | The blocks | 4 |
| 4.1 | General case | 4 |
| 4.2 | The mono-column blocks | 5 |
| 4.3 | The mono-row blocks | 5 |
| 4.4 | The mono-cell blocks | 5 |
| 4.5 | A small remark | 6 |
| 5 | The rules | 6 |
| 5.1 | Some differences with the classical environments | 7 |
| 5.1.1 | The vertical rules | 7 |
| 5.1.2 | The command <code>\cline</code> | 7 |
| 5.2 | The thickness and the color of the rules | 8 |
| 5.3 | The keys <code>hlines</code> and <code>vlines</code> | 8 |
| 5.4 | The key <code>hvlines</code> | 8 |
| 5.5 | The key <code>hvlines-except-corners</code> | 9 |
| 5.6 | The command <code>\diagbox</code> | 9 |
| 5.7 | Dotted rules | 10 |
| 6 | The color of the rows and columns | 10 |
| 6.1 | Use of <code>colortbl</code> | 10 |
| 6.2 | The tools of <code>nicematrix</code> in the code-before | 11 |
| 6.3 | Color tools with the syntax of <code>colortbl</code> | 13 |
| 7 | The width of the columns | 14 |
| 8 | The exterior rows and columns | 15 |
| 9 | The continuous dotted lines | 16 |
| 9.1 | The option <code>nullify-dots</code> | 18 |
| 9.2 | The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code> | 18 |
| 9.3 | How to generate the continuous dotted lines transparently | 19 |
| 9.4 | The labels of the dotted lines | 20 |
| 9.5 | Customization of the dotted lines | 20 |
| 9.6 | The dotted lines and the rules | 21 |
| 10 | The code-after | 21 |
| 11 | The notes in the tabulars | 22 |
| 11.1 | The footnotes | 22 |
| 11.2 | The notes of tabular | 22 |
| 11.3 | Customisation of the tabular notes | 24 |
| 11.4 | Use of <code>{NiceTabular}</code> with <code>threeparttable</code> | 25 |

| | | |
|-----------|--|------------|
| 12 | Other features | 26 |
| 12.1 | Use of the column type S of siunitx | 26 |
| 12.2 | Alignment option in {NiceMatrix} | 26 |
| 12.3 | The command \rotate | 26 |
| 12.4 | The option small | 27 |
| 12.5 | The counters iRow and jCol | 27 |
| 12.6 | The option light-syntax | 28 |
| 12.7 | Color of the delimiters | 28 |
| 12.8 | The environment {NiceArrayWithDelims} | 28 |
| 13 | Use of Tikz with nicematrix | 29 |
| 13.1 | The nodes corresponding to the contents of the cells | 29 |
| 13.2 | The “medium nodes” and the “large nodes” | 30 |
| 13.3 | The “row-nodes” and the “col-nodes” | 31 |
| 14 | API for the developpers | 32 |
| 15 | Technical remarks | 33 |
| 15.1 | Definition of new column types | 33 |
| 15.2 | Diagonal lines | 33 |
| 15.3 | The “empty” cells | 34 |
| 15.4 | The option exterior-arraycolsep | 34 |
| 15.5 | Incompatibilities | 34 |
| 16 | Examples | 35 |
| 16.1 | Notes in the tabulars | 35 |
| 16.2 | Dotted lines | 36 |
| 16.3 | Dotted lines which are no longer dotted | 38 |
| 16.4 | Width of the columns | 38 |
| 16.5 | How to highlight cells of a matrix | 39 |
| 16.6 | Direct use of the Tikz nodes | 41 |
| 17 | Implementation | 43 |
| 18 | History | 169 |
| | Index | 175 |