

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

June 30, 2020

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.

$$\begin{array}{c} L_1 \\ L_2 \\ \vdots \\ L_n \end{array} \begin{array}{c} C_1 \\ C_2 \dots \dots C_n \end{array} \left[\begin{array}{ccc} a_{11} & a_{12} \dots \dots a_{1n} \\ a_{21} & a_{22} \dots \dots a_{2n} \\ \vdots & \vdots \ddots \vdots \\ a_{n1} & a_{n2} \dots \dots a_{nn} \end{array} \right]$$

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).

This package requires and **loads** the packages `l3keys2e`, `xparse`, `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).

*This document corresponds to the version 4.4 of `nicematrix`, at the date of 2020/06/30.

1 The environments of this package

The package `nicematrix` defines the following new environments.

<code>{NiceTabular}</code>	<code>{NiceArray}</code>	<code>{NiceMatrix}</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}` and `{NiceTabular}` are similar to the environments `{array}` and `{tabular}` of the package `array` (which is loaded by `nicematrix`).

However, there are some small differences:

- For technical reasons, in the preamble of these environments, the user must use the letters `L`, `C` and `R`¹ instead of `l`, `c` and `r`, included in the commands `\multicolumn` and in the types of columns defined by `\newcolumntype`.
- * In `{NiceArray}` (and its variants), the columns of type `w` (ex. : `wc{1cm}`) are composed in math mode whereas, in `{array}` of `array`, they are composed in text mode.

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.**

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`. 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`.

```

\NiceMatrixOptions{cell-space-top-limit = 1pt,cell-space-bottom-limit = 1pt}


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


```

¹The column types `L`, `C` and `R` are defined locally inside `{NiceTabular}` or `{NiceArray}` with `\newcolumntype` of `array`. This definition overrides an eventual previous definition.

3 The vertical position of the arrays

The package `nicematrix` provides a option `baseline` for the vertical position of the arrays. This option takes 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`²).

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

1. an item

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

4 The blocks

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` don't create space by itself.

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. The second argument is the content of the block.

In `{NiceTabular}` the content of the block is composed in text mode. In the other environments, it is composed in math mode.

²It's also possible to use `\firsthline` in the environments of `nicematrix`.

```

\begin{NiceTabular}{CCCC}
rose      & tulipe & marguerite & dahlia \\
violette  & \Block{2-2}{\LARGE\color{blue} fleurs} & & souci \\
pervenche & & & lys \\
arum      & iris & jacinthe & muguet
\end{NiceTabular}

```

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

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

```

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

```

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

It’s also possible to use 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.

```

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

```

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

5 The rules

The usual techniques for the rules may be used in the environments of `nicematrix` (excepted `\vline`).

5.1 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 23.

5.2 A remark about `\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.

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

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

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

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

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

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

5.3 The keys `hlines` and `vlines`

The key `hlines` draws all the horizontal rules and the key `vlines` draws all the vertical rules. In fact, in the environments with delimiters (as `{pNiceMatrix}` or `{bNiceArray}`) the exterior rules are not drawn (as expected).

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

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

However, there is a difference between the key `vlines` and the use of the specifier “|” in the preamble of the environment: the rules drawn by `vlines` completely cross the double-rules drawn by `\hline\hline` (you don’t need `hhline`).

```
$\begin{NiceMatrix}[vlines] \hline
a & b & c & d \\ \hline \hline
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\ \hline
\end{NiceMatrix}$
```

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

If you use `booktabs` (which provides `\toprule`, `\midrule`, `\bottomrule`, etc.) and you really want to draw vertical rules (something opposed to the spirit of `booktabs`), you should remark that the key `vlines` is compatible with `booktabs`.

```
$\begin{NiceMatrix}[vlines] \toprule
a & b & c & d \\ \midrule
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\ \bottomrule
\end{NiceMatrix}$
```

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

5.4 The key `hvlines`

The key `hvlines` draws all the vertical and horizontal rules *excepted in the blocks*.³

```
\setlength{\arrayrulewidth}{1pt}
\begin{NiceTabular}{CCCC}[hvlines, rules/color=blue]
rose      & tulipe & marguerite & dahlia \\
violette  & \Block{2-2}{\LARGE\color{blue} 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 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}
```

³In fact, when the key `hvlines` (or the key `hvlines-except-corners` described just after) is in force, the rules are also not drawn in the virtual blocks delimited by cells relied par dotted lines (cf. p. 17).

				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.

```
\begin{NiceTabular}{*{6}{C}}%
  [hvlines-except-corners, 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.⁴

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

$\begin{smallmatrix} y \\ x \end{smallmatrix}$	<i>e</i>	<i>a</i>	<i>b</i>	<i>c</i>
<i>e</i>	<i>e</i>	<i>a</i>	<i>b</i>	<i>c</i>
<i>a</i>	<i>a</i>	<i>e</i>	<i>c</i>	<i>b</i>
<i>b</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>a</i>
<i>c</i>	<i>c</i>	<i>b</i>	<i>a</i>	<i>e</i>

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}
```

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

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

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

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

⁴The author of this document considers that type of construction as graphically poor.

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 rule⁵. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

6 The color of the rows and columns

With the classical package `colortbl`, it's possible to color the cells, rows and columns of a tabular. However, the resulting PDF is not always perfectly displayed by the PDF viewers, in particular in conjunction with rules. With some PDF viewers, some vertical rules seem to vanish. On the other side, some thin horizontal white lines may appear in some circumstances.

The package `nicematrix` provides similar tools which do not present these drawbacks. It provides a key `code-before`⁶ for some code which will be executed *before* the drawing of the tabular. In this `code-before`, new commands are available: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors` and `\chessboardcolors`.

These commands are independent of `colortbl`.⁷

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 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).

⁵In fact, this is true only for `\hline` and “|” but not for `\cline`.

⁶There is also a key `code-after` : see p. 17.

⁷Thus, it's possible to color the rules, the cells, the rows, the columns, etc. without loading `colortbl`.

```

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

```

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`⁸. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows of the tabular, beginning with the row whose number is given in first (mandatory) argument. The two other (mandatory) arguments are the colors.

```

\begin{NiceTabular}{LR}[hlines,code-before = \rowcolors{1}{blue!10}{}]
John & 12 \\
Stephen & 8 \\
Sarah & 18 \\
Ashley & 20 \\
Henry & 14 \\
Madison & 15 \\
\end{NiceTabular}

```

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

- The command `\chessboardcolors` takes in mandatory arguments two colors and colors the cells of the tabular in quincunx with these colors.

```

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

```

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

We have used the key `R` which aligns all the columns rightwards (cf. p. 18).

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

⁸The command `\rowcolors` of `color` is available when `xcolor` is loaded with the option `table`.

```

\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{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.

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`. In `{NiceTabular}`, the cells of such columns are composed in `text` mode but, in `{NiceArray}`, `{pNiceArray}`, etc., they are composed in `math` mode (whereas, in `{array}` of `array`, they are composed in `text` mode).

```

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

```

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

In the environments of `nicematrix`, it's also possible to fix the *minimal* width of all the columns 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.⁹

```

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

```

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

⁹The result is achieved with only one compilation (but Tikz will have written informations in the `.aux` file and a message requiring a second compilation will appear).

Without surprise, it's possible to fix the minimal width of the columns of all the 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`¹⁰. The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 3).

```
\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}
```

$$\begin{bmatrix} 9 & 17 \\ -2 & 5 \end{bmatrix} \quad \begin{bmatrix} 1 & 1245345 \\ 345 & 2 \end{bmatrix}$$

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

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

The dotted lines have been drawn with the tools presented p. 13.

We have several remarks to do.

¹⁰At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

- 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. 20) 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}}
$\begin{pNiceArray}{CC|CC}[first-row,last-row=5,first-col,last-col,nullify-dots]
    & C_1 & & \Cdots & & C_4 & & \\
L_1 & & a_{11} & & a_{12} & & a_{13} & & a_{14} & & L_1 \\
\vdots & & a_{21} & & a_{22} & & a_{23} & & a_{24} & & \vdots \\
\hline
    & & a_{31} & & a_{32} & & a_{33} & & a_{34} & & \\
L_4 & & a_{41} & & a_{42} & & a_{43} & & a_{44} & & L_4 \\
    & & C_1 & & \Cdots & & C_4 & & \\
\end{pNiceArray}$
```

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

Remarks

- As shown in the previous example, the horizontal and 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 23.
- 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. 10) 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`.¹¹

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells¹² on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it's an horizontal line; for `\Vdots`, it's a vertical line and for `\Ddots` and `\Iddots` diagonal ones. It's possible to change the color of these lines with the option `color`.¹³

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

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

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

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

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

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

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

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

In the first column of this exemple, there are two instructions `\Vdots` but only one dotted line is drawn (there is no overlapping graphic objects in the resulting PDF¹⁴).

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

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

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

¹¹The 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`.

¹²The precise definition of a “non-empty cell” is given below (cf. p. 24).

¹³It's also possible to change the color of all these dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 16.

¹⁴And it's not possible to draw a `\Ldots` and a `\Cdots` line between the same cells.

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\l` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.¹⁵

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

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

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

9.1 The option `nullify-dots`

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

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

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

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

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

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

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

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

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

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).

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

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

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

9.2 The command `\Hdotsfor`

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.

¹⁵In `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. 10

```

 $\begin{pNiceMatrix}$ 
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`).

9.3 How to generate the continuous dotted lines transparently

The package `nicematrix` provides an option called `transparent` for using existing code transparently in the environments of the `amsmath`: `{matrix}`, `{pmatrix}`, `{bmatrix}`, etc. In fact, this option is an alias for the conjunction of two options: `renew-dots` and `renew-matrix`.¹⁶

- The option `renew-dots`
With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`¹¹ 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 option `transparent`, a classical code gives directly the output of `nicematrix`.

```

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

```

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

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. 17) 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
 $\end{bNiceMatrix}$ 

```

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

¹⁶The options `renew-dots`, `renew-matrix` and `transparent` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage` (it's an exception for these three specific options.)

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. 17) 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. 11.

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).¹⁷

`\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}
```

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

¹⁷The 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.

9.6 The dotted lines and the key hvlines

We have said (cf. p. 6) that the key `hvlines` draws all the horizontal and vertical rules, excepted in the blocks. In fact, when this key is in force, the rules are also not drawn in the virtual blocks delimited by cells relied par dotted lines.

```
\NiceMatrixOptions{nullify-dots}
$\begin{pNiceMatrix}[rules/color=gray,hvlines,margin]
0      & \Cdots & & & & 0      & \\
1      & \Cdots & & & 1      & 2      & \\
0      & \Ddots & & & \Vdots & \Vdots & \\
\Vdots & \Ddots & & & & & \\
      & & & & & & \\
0      & \Cdots & & 0 & 1      & 2      & \\
\end{pNiceMatrix}$
```

$$\left(\begin{array}{cccccc|c} 0 & \cdots & \cdots & \cdots & \cdots & 0 & \\ \hline 1 & \cdots & \cdots & \cdots & \cdots & 1 & 2 \\ 0 & \ddots & & & & \vdots & \\ \vdots & & \ddots & & & \vdots & \\ \vdots & & & \ddots & & \vdots & \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 2 \end{array} \right)$$

10 The code-after

The option `code-after` may be used to give some code that will be excuted after the construction of the matrix.¹⁸

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 adjacents 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}
```

$$\left(\begin{array}{cccccc} I & 0 & \cdots & \cdots & 0 \\ 0 & I & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & I & 0 \\ 0 & \cdots & \cdots & 0 & I \end{array} \right)$$

For the readability 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. 28.

11 Other features

11.1 Use of the column type S of siunitx

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

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

$$\left(\begin{array}{cccccc} C_1 & \cdots & \cdots & \cdots & \cdots & C_n \\ 2.3 & 0 & \cdots & \cdots & \cdots & 0 \\ 12.4 & \vdots & & & & \vdots \\ 1.45 & \vdots & & & & \vdots \\ 7.2 & 0 & \cdots & \cdots & \cdots & 0 \end{array} \right)$$

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

¹⁸There is also a key `code-before` described p. 8.

11.2 Aligment option in {NiceMatrix}

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

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

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

There is also a key `S` which sets all the columns all type `S` of `siunitx` (if this package is loaded).¹⁹

11.3 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`.

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

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

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

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

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

11.4 The option `small`

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

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

¹⁹This is a part of the functionality provided by the environments `{pmatrix*}`, `{bmatrix*}`, etc. of `mathtools`.

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

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.

11.5 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 column²⁰. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `code-before` (cf. p. 8) and in the `code-after` (cf. p. 17), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

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

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

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

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

These commands take 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 eventual exterior rows and columns).

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

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

²⁰We 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.

11.6 The option `light-syntax`

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

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

The following example has been composed with XeLaTeX with `unicode-math`, which allows the use of greek letters directly in the TeX source.

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

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

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

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.²¹

11.7 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}$
```

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

12 Utilisation of Tikz with `nicematrix`

12.1 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).

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.

²¹The 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.

```

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

```

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

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

In the **code-after**, and if Tikz is loaded, the things are easier. One may design the nodes with the form *i-j*: there is no need to indicate the environment which is of course the current environment.

```

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

```

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

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 28).

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

12.2 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**.²²

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

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

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

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.²³

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

The “large nodes” of the first column and last column may appear too small for some usage. That's why it's possible to use the options **left-margin** and **right-margin** to add space on both sides of

²²There is also an option **create-extra-nodes** which is an alias for the conjunction of **create-medium-nodes** and **create-large-nodes**.

²³There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 11).

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`.²⁴

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

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

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

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

Here is an array composed with the following code:

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

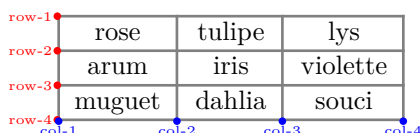
fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

12.3 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`.



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)`.

²⁴The 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.

```

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

```

```

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

```

13 Technical remarks

13.1 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 eventual 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 job²⁵:

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

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

```

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

```

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

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

²⁵The command `\vrule` is a TeX (and not LaTeX) command.

13.2 Diagonal lines

By default, all the diagonal lines²⁶ of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That’s why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

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

Example with parallelization (default):

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

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

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

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

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

The same example without parallelization:

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

13.3 The “empty” cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cells 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 `&`). Indeed, a cell which only contains `\hspace*{1cm}` may be considered as empty.

For `nicematrix`, 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 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`.

²⁶We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

13.4 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 idea²⁷. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`²⁸. 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).

13.5 Incompatibilities

The package `nicematrix` is not compatible with `threeparttable`.

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

14 Examples

14.1 Dotted lines

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

```


$$\begin{array}{ccccccc} 0 & & 1 & & 0 & & \\ \vdots & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ 0 & & 0 & & & & 1 \\ 1 & & 0 & & & & 0 \end{array}$$


```

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

```


$$\begin{array}{ccccccc} 1 & & & & & & 1 \\ \vdots & & & & & & 0 \\ & & \iddots & & \iddots & & \\ 1 & & 0 & & & & 0 \end{array}$$


```

²⁷In 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).*

²⁸And 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

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}
```

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

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}
```

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

An example for the resultant of two polynomials:

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

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

An example for a linear system:

```

 $\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 0 & \\ 0 & 1 & 0 & \cdots & 0 & & L_2 - L_1 \\ 0 & 0 & 1 & \ddots & \vdots & & L_3 - L_1 \\ & & & \ddots & \vdots & \vdots & \\ \vdots & & & \ddots & 0 & & \\ 0 & & & \cdots & 0 & 1 & 0 & L_n - L_1 \end{pmatrix}$ 

```

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

14.2 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}}
 $\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \ddots & \vdots \\ \vdots & & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix}$ 

```

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

14.3 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}

```

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

14.4 How to highlight cells of the matrix

The following examples 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}

```

In order to highlight a cell of a matrix, it's possible to “draw” one of the correspondent nodes (the “normal node”, the “medium node” or the “large node”). In the following example, we use the “large nodes” of the diagonal of the matrix (with the Tikz key “`name suffix`”, it's easy to use the “large nodes”).

We redraw the nodes with other nodes by using the Tikz library `fit`. Since we want to redraw the nodes exactly, we have to set `inner sep = 0 pt` (if we don't do that, the new nodes will be larger than the nodes created by `nicematrix`).

```

$\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
\begin{tikzpicture}[name suffix = -large,
    every node/.style = {draw,inner sep = 0 pt}]
    \node [fit = (1-1)] {} ;
    \node [fit = (2-2)] {} ;
    \node [fit = (3-3)] {} ;
    \node [fit = (4-4)] {} ;
\end{tikzpicture}
\end{pNiceArray}$

```

$$\left(\begin{array}{|c|c|c|c|} \hline a_{11} & a_{12} & a_{13} & a_{14} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} \\ \hline \end{array} \right)$$

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.²⁹

It's possible to color a row with `\rowcolor` in the *code-before* (or with `\rowcolor` of `colortbl` in the first cell of the row). However, it's not possible to do a fine tuning. That's why we describe now 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`.

```

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

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

```

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

This code fails with `latex-dvips-ps2pdf` because Tikz for `dvips`, as for now, doesn't support blend modes. However, the following code, in the preamble, should activate blend modes in this way of compilation.

```

\ExplSyntaxOn
\makeatletter
\tl_set:Nn \l_tmpa_tl {pgfsys-dvips.def}
\tl_if_eq:NNT \l_tmpa_tl \pgfsysdriver
{ \cs_set:Npn \pgfsys@blend@mode#1{\special{ps:~/\tl_upper_case:n #1~.setblendmode}}}
\makeatother
\ExplSyntaxOff

```

²⁹For the command `\cline`, see the remark p. 5.

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

```

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

```

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

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

```

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

```

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

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

```

\tikzset{mes-options/.style={remember picture,
                             overlay,
                             name prefix = example-,
                             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}
```

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

14.5 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}
& \\
& \end{array}
```

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

```
\begin{bNiceArray}>{\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}

```

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

15 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   {Mathematical matrices with 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_@@_booktabs_loaded_bool
22 \bool_new:N \c_@@_tikz_loaded_bool
23 \AtBeginDocument
24 {
25   \ifpackageloaded { booktabs }
26     { \bool_set_true:N \c_@@_booktabs_loaded_bool }
27     { }
28   \ifpackageloaded { tikz }
29     {
```

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.

```
30   \bool_set_true:N \c_@@_tikz_loaded_bool
31   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
32   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
```

```

33     }
34     {
35         \tl_const:Nn \c_@@pgfortikzpicture_tl { \exp_not:N \pgfpicture }
36         \tl_const:Nn \c_@@endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
37     }
38 }

```

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.

```

39 \bool_new:N \c_@@revtex_bool
40 \ifclassloaded { revtex4-1 }
41 { \bool_set_true:N \c_@@revtex_bool }
42 { }
43 \ifclassloaded { revtex4-2 }
44 { \bool_set_true:N \c_@@revtex_bool }
45 { }

```

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

```

46 \ProvideDocumentCommand \iddots { }
47 {
48     \mathinner
49     {
50         \tex_mkern:D 1 mu
51         \box_move_up:nn { 1 pt } { \hbox:n { . } }
52         \tex_mkern:D 2 mu
53         \box_move_up:nn { 4 pt } { \hbox:n { . } }
54         \tex_mkern:D 2 mu
55         \box_move_up:nn { 7 pt }
56         { \vbox:n { \kern 7 pt \hbox:n { . } } }
57         \tex_mkern:D 1 mu
58     }
59 }

```

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.

```

60 \AtBeginDocument
61 {
62     \ifpackageloaded { booktabs }
63     { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
64     { }
65 }
66 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
67 {
68     \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`).

```

69     \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
70     {
71         \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
72         { \@@_old_pgful@check@rerun { ##1 } { ##2 } }
73     }
74 }

```

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

```

75 \bool_new:N \c_@@colortbl_loaded_bool
76 \AtBeginDocument

```

```

77 {
78   \@ifpackageloaded { colortbl }
79   { \bool_set_true:N \c_@@_colortbl_loaded_bool }
80   {

```

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.

```

81   \cs_set_protected:Npn \CT@arc@ { }
82   \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
83   \cs_set:Npn \CT@arc@ #1 #2
84   {
85     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
86     { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
87   }
88   \cs_set:Npn \hline
89   {
90     \noalign { \ifnum 0 = ` } \fi
91     \cs_set_eq:NN \hskip \vskip
92     \cs_set_eq:NN \vrule \hrule
93     \cs_set_eq:NN \@width \@height
94     { \CT@arc@ \vline }
95     \futurelet \reserved@a
96     \@xhline
97   }
98 }
99 }

```

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.

```

100 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
101 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
102 {
103   \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
104   \int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
105   \multispan { \int_eval:n { #2 - #1 + 1 } }
106   { \CT@arc@ \leaders \hrule \@height \arrayrulewidth \hfill }

```

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`.

```

107   \everycr { }
108   \cr
109   \noalign { \skip_vertical:N -\arrayrulewidth }
110 }

```

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

```

111 \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`.

```

112 { \@@_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*.

```

113 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1-#2 \q_stop }
114 \cs_set:Npn \@@_cline_i:w #1-#2-#3 \q_stop
115 {

```

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).

```

116   \int_compare:nNnT { #1 } < { #2 }
117   { \multispan { \int_eval:n { #2 - #1 } } & }
118   \multispan { \int_eval:n { #3 - #2 + 1 } }
119   { \CT@arc@ \leaders \hrule \@height \arrayrulewidth \hfill }

```

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

```

120 \peek_meaning_remove_ignore_spaces:NTF \cline
121 { & \@@_cline_i:en { \@@_succ:n { #3 } } }
122 { \everycr { } \cr }
123 }
124 \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.

```

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

```

The following command is a small shortcut.

```

127 \cs_new:Npn \@@_math_toggle_token:
128 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

129 \cs_new_protected:Npn \@@_set_CT@arc@:
130 { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
131 \cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
132 { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
133 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
134 { \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`.

```

135 \bool_new:N \c_@@_siunitx_loaded_bool
136 \AtBeginDocument
137 {
138   \@ifpackageloaded { siunitx }
139   { \bool_set_true:N \c_@@_siunitx_loaded_bool }
140   { }
141 }

```

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} }
    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:`.

```

142 \cs_set_protected:Npn \@@_adapt_S_column:
143 {
144   \bool_if:NT \c_@@_siunitx_loaded_bool
145   {
146     \group_begin:
147     \@temptokena = { }

```

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

```

148   \cs_set_eq:NN \NC@find \prg_do_nothing:
149   \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`).

```

150   \tl_gset:NV \g_tmpa_tl \@temptokena
151   \group_end:
152   \tl_new:N \c_@@_table_collect_begin_tl
153   \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
154   \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
155   \tl_new:N \c_@@_table_print_tl
156   \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).

```

157   \cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
158 }
159 }

```

The command `\@@_renew_NC@rewrite@S:` will be used in each environment of `nicematrix` in order to “rewrite” the `S` column in each environment (only if the boolean `\c_@@_siunitx_loaded_bool` is raised, of course).

```

160 \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
161 {
162   \renewcommand*{\NC@rewrite@S}[1] []
163   {
164     \@temptokena \exp_after:wN
165     {
166       \tex_the:D \@temptokena
167       > { \@@_Cell: \c_@@_table_collect_begin_tl S {##1} }
168       c
169       < { \c_@@_table_print_tl \@@_end_Cell: }
170     }
171     \NC@find
172   }
173 }

```

Parameters

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

```

174 \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).

```
175 \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.

```
176 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
177 { \int_use:N \g_@@_env_int }
```

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

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

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

```
180 \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).

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

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

```
182 \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.

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

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

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

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

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

```
186 \cs_new_protected:Npn \@@_test_if_math_mode:
187 {
188   \if_mode_math: \else:
189     \@@_fatal:n { Outside~math~mode }
190   \fi:
191 }
```

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

```
192 \colorlet { nicematrix-last-col } { . }
193 \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*).

```
194 \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*.

```
195 \str_new:N \g_@@_com_or_env_str
196 \str_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.

```
197 \cs_new:Npn \@@_full_name_env:
198 {
199   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
200   { command \space \c_backslash_str \g_@@_name_env_str }
201   { environment \space \{ \g_@@_name_env_str \} }
202 }
```

The following token list corresponds to the option `code-after` (it's also possible to set the value of that parameter with the command `\CodeAfter`).

```
203 \tl_new:N \g_@@_code_after_tl
```

The following token list has a function similar to `\g_@@_code_after_tl` but it is used internally by `nicematrix`. In fact, we have to distinguish between `\g_@@_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.

```
204 \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.

```
205 \int_new:N \l_@@_old_iRow_int
206 \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.

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

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).

```
208 \bool_new:N \g_@@_row_of_col_done_bool
```

The following flag will be raised when the key `code-before` is used in the environment. Indeed, if there is a `code-before` in the environment, we will manage to have the `row` nodes and the `col` nodes available *before* the creation of the array.

```
209 \bool_new:N \l_@@_code_before_bool
```

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

```
210 \dim_new:N \l_@@_x_initial_dim
211 \dim_new:N \l_@@_y_initial_dim
212 \dim_new:N \l_@@_x_final_dim
213 \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`).

```
214 \dim_zero_new:N \l_tmpc_dim
215 \dim_zero_new:N \l_tmpd_dim
```

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

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

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

```
217 \dim_new:N \g_@@_old_arraycolsep_dim
```

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

```
218 \dim_new:N \g_@@_width_last_col_dim
219 \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.

```
220 \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}`.

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

The sequence `\g_@@_pos_of_blocks_seq` will be used when we will draw the rules required by the keys `hvlines` or `hvlines-except-corners`.

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}`.

```
222 \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).

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.

```
223 \int_new:N \l_@@_first_row_int
224 \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.

```
225 \int_new:N \l_@@_first_col_int
226 \int_set:Nn \l_@@_first_col_int 1
```

- **Last row**

The counter `\l_@@_last_row_int` is the number of the eventual “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).

```
227 \int_new:N \l_@@_last_row_int
228 \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”.³⁰

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

Idem for `\l_@@_last_col_without_value_bool`

```
230 \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 there is a last column but we don’t know its value because the user has used the option `last-col` without value (it’s possible in an environment without preamble like `{pNiceMatrix}`). A value of `0` means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`).

```
231 \int_new:N \l_@@_last_col_int
232 \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.

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

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

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.

```
234 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
235 {
236   \begin { pgfscope }
237   \pgfset
238   {
239     outer-sep = \c_zero_dim ,
```

³⁰We 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.

```

240     inner~sep = \c_zero_dim ,
241     minimum~size = \c_zero_dim
242 }
243 \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
244 \pgfnode
245 { rectangle }
246 { center }
247 {
248     \vbox_to_ht:nn
249     { \dim_abs:n { #5 - #3 } }
250     {
251         \vfill
252         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
253     }
254 }
255 { #1 }
256 { }
257 \end { pgfscope }
258 }

```

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.

```

259 \cs_new_protected:Npn \@@pgf_rect_node:nnn #1 #2 #3
260 {
261     \begin { pgfscope }
262     \pgfset
263     {
264         outer~sep = \c_zero_dim ,
265         inner~sep = \c_zero_dim ,
266         minimum~size = \c_zero_dim
267     }
268     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
269     \pgfpointdiff { #3 } { #2 }
270     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
271     \pgfnode
272     { rectangle }
273     { center }
274     {
275         \vbox_to_ht:nn
276         { \dim_abs:n \l_tmpb_dim }
277         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
278     }
279     { #1 }
280     { }
281     \end { pgfscope }
282 }

```

The options

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`.

```

283 \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`).

```

284 \dim_new:N \l_@@_cell_space_top_limit_dim
285 \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.

```
286 \dim_new:N \l_@@_inter_dots_dim
287 \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).

```
288 \dim_new:N \l_@@_xdots_shorten_dim
289 \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.

```
290 \dim_new:N \l_@@_radius_dim
291 \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.

```
292 \tl_new:N \l_@@_xdots_line_style_tl
293 \tl_const:Nn \c_@@_standard_tl { standard }
294 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
295 \bool_new:N \l_@@_light_syntax_bool
```

The string `\l_@@_baseline_str` 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).

```
296 \str_new:N \l_@@_baseline_str
297 \str_set:Nn \l_@@_baseline_str 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`).

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

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

```
299 \bool_new:N \l_@@_parallelize_diags_bool
300 \bool_set_true:N \l_@@_parallelize_diags_bool
```

The flag `\l_@@_hlines_bool` corresponds to the key `hlines`, the flag `\l_@@_vlines_bool` to the key `vlines` and the flag `hvlines` to the key `hvlines`. Since version 4.1, the key `hvlines` is no longer a mere alias for the conjunction of `hlines` and `vlines`. Indeed, with `hvlines`, the vertical and horizontal rules are *not* drawn within the blocks (created by `\Block`).

```
301 \bool_new:N \l_@@_hlines_bool
302 \bool_new:N \l_@@_vlines_bool
303 \bool_new:N \l_@@_hvlines_bool
```

The flag `\l_@@_hlines_except_corners_bool` corresponds to the key `hlines-except-corners`.

```
304 \bool_new:N \l_@@_hvlines_except_corners_bool
```

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.).

```
305 \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).

```
306 \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.

```
307 \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”.

```
308 \bool_new:N \l_@@_medium_nodes_bool
```

```
309 \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`).

```
310 \dim_new:N \l_@@_left_margin_dim
```

```
311 \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`.

```
312 \dim_new:N \l_@@_extra_left_margin_dim
```

```
313 \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.

```
314 \tl_new:N \l_@@_end_of_row_tl
```

```
315 \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 “:”.

```
316 \tl_new:N \l_@@_xdots_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.

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

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

```
318 \keys_define:nn { NiceMatrix / xdots }
```

```
319 {
```

```
320   line-style .code:n =
```

```
321   {
```

```
322     \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.

```

323     { \cs_if_exist_p:N \tikzpicture }
324     { \str_if_eq_p:nn { #1 } { standard } }
325     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
326     { \@@_error:n { bad-option-for-line-style } }
327   } ,
328   line-style .value_required:n = true ,
329   color .tl_set:N = \l_@@_xdots_color_tl ,
330   color .value_required:n = true ,
331   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
332   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 `_`.

```

333   down .tl_set:N = \l_@@_xdots_down_tl ,
334   up .tl_set:N = \l_@@_xdots_up_tl ,
335   unknown .code:n = \@@_error:n { Unknown-option-for-~xdots }
336 }

337 \keys_define:nn { NiceMatrix / rules }
338 {
339   color .tl_set:N = \l_@@_rules_color_tl ,
340   color .value_required:n = true ,
341   width .dim_set:N = \arrayrulewidth ,
342   width .value_required:n = true
343 }

344 \keys_define:nn { NiceMatrix / Global }
345 {
346   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
347   standard-cline .default:n = true ,
348   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
349   cell-space-top-limit .value_required:n = true ,
350   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
351   cell-space-bottom-limit .value_required:n = true ,
352   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
353   max-delimiter-width .bool_set:N = \l_@@_max_delimiter_width_bool ,
354   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
355   light-syntax .default:n = true ,
356   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
357   end-of-row .value_required:n = true ,
358   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
359   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
360   last-row .int_set:N = \l_@@_last_row_int ,
361   last-row .default:n = -1 ,
362   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
363   code-for-first-col .value_required:n = true ,
364   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
365   code-for-last-col .value_required:n = true ,
366   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
367   code-for-first-row .value_required:n = true ,
368   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
369   code-for-last-row .value_required:n = true ,
370   hlines .bool_set:N = \l_@@_hlines_bool ,
371   vlines .bool_set:N = \l_@@_vlines_bool ,
372   hvlines .code:n =
373   {
374     \bool_set_true:N \l_@@_hvlines_bool
375     \bool_set_true:N \l_@@_vlines_bool
376     \bool_set_true:N \l_@@_hlines_bool
377   } ,
378   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.

```

379   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
380   renew-dots .value_forbidden:n = true ,
381   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
382   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
383   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
384   create-extra-nodes .meta:n =
385     { create-medium-nodes , create-large-nodes } ,
386   left-margin .dim_set:N = \l_@@_left_margin_dim ,
387   left-margin .default:n = \arraycolsep ,
388   right-margin .dim_set:N = \l_@@_right_margin_dim ,
389   right-margin .default:n = \arraycolsep ,
390   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
391   margin .default:n = \arraycolsep ,
392   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
393   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
394   extra-margin .meta:n =
395     { extra-left-margin = #1 , extra-right-margin = #1 } ,
396   extra-margin .value_required:n = true
397 }

```

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

```

398 \keys_define:nn { NiceMatrix / Env }
399 {
400   hvlines-except-corners .bool_set:N = \l_@@_hvlines_except_corners_bool ,
401   hvlines-except-corners .default:n = true ,
402   code-before .code:n =
403     {
404       \tl_if_empty:nF { #1 }
405       {
406         \tl_set:Nn \l_@@_code_before_tl { #1 }
407         \bool_set_true:N \l_@@_code_before_bool
408       }
409     } ,

```

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

```

410   c .code:n = \str_set:Nn \l_@@_baseline_str c ,
411   t .code:n = \str_set:Nn \l_@@_baseline_str t ,
412   b .code:n = \str_set:Nn \l_@@_baseline_str b ,
413   baseline .tl_set:N = \l_@@_baseline_str ,
414   baseline .value_required:n = true ,
415   columns-width .code:n =
416     \str_if_eq:nnTF { #1 } { auto }
417     { \bool_set_true:N \l_@@_auto_columns_width_bool }
418     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
419   columns-width .value_required:n = true ,
420   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.

```

421   \legacy_if:nF { measuring@ }
422   {
423     \str_set:Nn \l_tmpa_str { #1 }
424     \seq_if_in:NVTf \g_@@_names_seq \l_tmpa_str
425     { \@@_error:nn { Duplicate-name } { #1 } }
426     { \seq_gput_left:Nv \g_@@_names_seq \l_tmpa_str }
427     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
428   } ,
429   name .value_required:n = true ,

```

```

430   code-after .tl_gset:N = \g_@@_code_after_tl ,
431   code-after .value_required:n = true ,
432 }

```

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

```

433 \keys_define:nn { NiceMatrix }
434 {
435   NiceMatrixOptions .inherit:n =
436   {
437     NiceMatrix / Global ,
438   } ,
439   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
440   NiceMatrix .inherit:n =
441   {
442     NiceMatrix / Global ,
443     NiceMatrix / Env ,
444   } ,
445   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
446   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
447   NiceTabular .inherit:n =
448   {
449     NiceMatrix / Global ,
450     NiceMatrix / Env
451   } ,
452   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
453   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
454   NiceArray .inherit:n =
455   {
456     NiceMatrix / Global ,
457     NiceMatrix / Env ,
458   } ,
459   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
460   NiceArray / rules .inherit:n = NiceMatrix / rules ,
461   pNiceArray .inherit:n =
462   {
463     NiceMatrix / Global ,
464     NiceMatrix / Env ,
465   } ,
466   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
467   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
468 }

```

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

```

469 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
470 {
471   last-col .code:n = \tl_if_empty:nF { #1 }
472   { \@@_error:n { last-col~non~empty~for~NiceMatrixOptions } }
473   \int_zero:N \l_@@_last_col_int ,
474   small .bool_set:N = \l_@@_small_bool ,
475   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.

```

476   renew-matrix .code:n = \@@_renew_matrix: ,
477   renew-matrix .value_forbidden:n = true ,
478   transparent .meta:n = { renew-dots , renew-matrix } ,
479   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}`.

```

480   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.

```

481 columns-width .code:n =
482   \str_if_eq:nnTF { #1 } { auto }
483   { \@@_error:n { Option~auto~for~columns~width } }
484   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

485 allow-duplicate-names .code:n =
486   \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
487 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`).

```

488 letter-for-dotted-lines .code:n =
489   {
490     \int_compare:nTF { \tl_count:n { #1 } = 1 }
491     { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
492     { \@@_error:n { Bad~value~for~letter~for~dotted~lines } }
493   } ,
494 letter-for-dotted-lines .value_required:n = true ,
495 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
496 }
497 \str_new:N \l_@@_letter_for_dotted_lines_str
498 \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.

```

499 \NewDocumentCommand \NiceMatrixOptions { m }
500   { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

501 \keys_define:nn { NiceMatrix / NiceMatrix }
502   {
503     last-col .code:n = \tl_if_empty:nTF {#1}
504       {
505         \bool_set_true:N \l_@@_last_col_without_value_bool
506         \int_set:Nn \l_@@_last_col_int { -1 }
507       }
508       { \int_set:Nn \l_@@_last_col_int { #1 } } ,
509     l .code:n = \tl_set:Nn \l_@@_type_of_col_tl L ,
510     r .code:n = \tl_set:Nn \l_@@_type_of_col_tl R ,
511     L .code:n = \tl_set:Nn \l_@@_type_of_col_tl L ,
512     R .code:n = \tl_set:Nn \l_@@_type_of_col_tl R ,
513     S .code:n = \bool_if:NTF \c_@@_siunitx_loaded_bool
514       { \tl_set:Nn \l_@@_type_of_col_tl S }
515       { \@@_error:n { option~S~without~siunitx } } ,
516     small .bool_set:N = \l_@@_small_bool ,
517     small .value_forbidden:n = true ,
518     unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
519   }

```

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

```
520 \keys_define:nn { NiceMatrix / NiceArray }
521 {
```

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.

```
522   small .bool_set:N = \l_@@_small_bool ,
523   small .value_forbidden:n = true ,
524   last-col .code:n = \tl_if_empty:nF { #1 }
525     { \@@_error:n { last-col~non-empty~for~NiceArray } }
526     \int_zero:N \l_@@_last_col_int ,
527   unknown .code:n = \@@_error:n { Unknown~option~for~NiceArray }
528 }

529 \keys_define:nn { NiceMatrix / pNiceArray }
530 {
531   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
532   last-col .code:n = \tl_if_empty:nF {#1}
533     { \@@_error:n { last-col~non-empty~for~NiceArray } }
534     \int_zero:N \l_@@_last_col_int ,
535   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
536   small .bool_set:N = \l_@@_small_bool ,
537   small .value_forbidden:n = true ,
538   unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
539 }
```

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

```
540 \keys_define:nn { NiceMatrix / NiceTabular }
541 {
542   last-col .code:n = \tl_if_empty:nF {#1}
543     { \@@_error:n { last-col~non-empty~for~NiceArray } }
544     \int_zero:N \l_@@_last_col_int ,
545   unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular }
546 }
```

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}).

```
547 \cs_new_protected:Npn \@@_Cell:
548 {
```

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

```
549   \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.

```
550   \int_compare:nNnT \c@jCol = 1
551     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
552   \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).

```
553   \hbox_set:Nw \l_@@_cell_box
554   \bool_if:NF \l_@@_NiceTabular_bool
```

```

555     {
556       \c_math_toggle_token
557       \bool_if:NT \l_@@_small_bool \scriptstyle
558     }

```

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.

```

559   \int_compare:nNnTF \c@iRow = 0
560   {
561     \int_compare:nNnT \c@jCol > 0
562     {
563       \l_@@_code_for_first_row_tl
564       \xglobal \colorlet { nicematrix-first-row } { . }
565     }
566   }
567   {
568     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
569     {
570       \l_@@_code_for_last_row_tl
571       \xglobal \colorlet { nicematrix-last-row } { . }
572     }
573   }
574 }

```

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.

```

575 \cs_new_protected:Npn \@@_begin_of_row:
576 {
577   \int_gincr:N \c@iRow
578   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
579   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
580   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
581   \pgfpicture
582   \pgfrememberpicturepositiononpagetrue
583   \pgfcoordinate
584   { \@@_env: - row - \int_use:N \c@iRow - base }
585   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
586   \str_if_empty:NF \l_@@_name_str
587   {
588     \pgfnodealias
589     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
590     { \@@_env: - row - \int_use:N \c@iRow - base }
591   }
592   \endpgfpicture
593 }

```

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 will be dynamically added to this command.

```

594 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
595 {
596   \int_compare:nNnTF \c@iRow = 0
597   {
598     \dim_gset:Nn \g_@@_dp_row_zero_dim
599     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
600     \dim_gset:Nn \g_@@_ht_row_zero_dim
601     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
602   }
603   {
604     \int_compare:nNnT \c@iRow = 1

```

```

605     {
606         \dim_gset:Nn \g_@@_ht_row_one_dim
607         { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
608     }
609 }
610 }

```

```

611 \cs_new_protected:Npn \@@_end_Cell:

```

```

612 {
613     \@@_math_toggle_token:
614     \hbox_set_end:
615     \box_set_ht:Nn \l_@@_cell_box
616     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
617     \box_set_dp:Nn \l_@@_cell_box
618     { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }

```

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

```

619     \dim_gset:Nn \g_@@_max_cell_width_dim
620     { \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”.

```

621     \@@_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 use 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.

```

622     \bool_if:NTF \g_@@_empty_cell_bool
623     { \box_use_drop:N \l_@@_cell_box }
624     {
625         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
626         \@@_node_for_the_cell:
627         { \box_use_drop:N \l_@@_cell_box }
628     }
629     \bool_gset_false:N \g_@@_empty_cell_bool
630 }

```

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

```

631 \cs_new_protected:Npn \@@_node_for_the_cell:
632 {
633     \pgfpicture
634     \pgfsetbaseline \c_zero_dim
635     \pgfrememberpicturepositiononpagetrue
636     \pgfset
637     {
638         inner~sep = \c_zero_dim ,
639         minimum~width = \c_zero_dim

```

```

640     }
641     \pgfnode
642     { rectangle }
643     { base }
644     { \box_use_drop:N \l_@@_cell_box }
645     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
646     { }
647     \str_if_empty:NF \l_@@_name_str
648     {
649         \pgfnodealias
650         { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
651         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
652     }
653     \endpgfpicture
654 }

```

The first argument of the following command `\@@_instruction_of_type:nn` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). The second 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}

```

```

655 \cs_new_protected:Npn \@@_instruction_of_type:nn #1 #2
656 {

```

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).

```

657     \tl_gput_right:cx
658     { g_@@_ #1 _ lines _ tl }
659     {
660         \use:c { @@ _ draw _ #1 : nnn }
661         { \int_use:N \c@iRow }
662         { \int_use:N \c@jCol }
663         { \exp_not:n { #2 } }
664     }
665 }

```

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`.

```

666 \cs_new_protected:Npn \@@_array:
667 {
668     \bool_if:NTF \c_@@_revtex_bool
669     {
670         \cs_set_eq:NN \@acol1 \@arrayacol
671         \cs_set_eq:NN \@acolr \@arrayacol
672         \cs_set_eq:NN \@acol \@arrayacol
673         \cs_set:Npn \@halignto { }
674         \@array@array
675     }
676     \array

```

`\l_@@_baseline_str` 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.

```
677 [ \str_if_eq:VnTF \l_@@_baseline_str c c t ]
678 }
```

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.

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

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

```
680 \cs_new_protected:Npn \@@_create_row_node:
681 {
```

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

```
682   \hbox
683   {
684     \bool_if:NT \l_@@_code_before_bool
685     {
686       \vtop
687       {
688         \skip_vertical:N 0.5\arrayrulewidth
689         \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
690         \skip_vertical:N -0.5\arrayrulewidth
691       }
692     }
693     \pgfpicture
694     \pgfrememberpicturepositiononpagetrue
695     \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
696     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
697     \str_if_empty:NF \l_@@_name_str
698     {
699       \pgfnodealias
700       { \l_@@_name_str - row - \int_use:N \c@iRow }
701       { \@@_env: - row - \int_use:N \c@iRow }
702     }
703     \endpgfpicture
704   }
705 }
```

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

```
706 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
707 \cs_new_protected:Npn \@@_everycr_i:
708 {
709   \int_gzero:N \c@jCol
710   \bool_if:NF \g_@@_row_of_col_done_bool
711   {
712     \@@_create_row_node:
```

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

```
713   \bool_if:NT \l_@@_hlines_bool
714   {
```

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.

```
715     \int_compare:nNnT \c@iRow > { -1 }
716     {
717       \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@`.

```
718     { \hrule height \arrayrulewidth width \c_zero_dim }
```

```

719     }
720   }
721 }
722 }

```

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`, `W`, `p`, `m` and `b`).

```

723 \cs_set_protected:Npn \@@_newcolumnntype #1
724 {
725   \cs_if_free:cT { NC @ find @ #1 }
726   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
727   \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
728   \peek_meaning:NTF [
729     { \newcol@ #1 }
730     { \newcol@ #1 [ 0 ] }
731   }

```

The following command will be used to redefine the column types `p`, `m` and `b`. That means that it will be used three times. The first argument is the letter of the column type (`p`, `m` or `b`). The second is the letter of position for the environment `{minipage}` (`t`, `c` or `b`).

```

732 \cs_new_protected:Npn \@@_define_columntype:nn #1 #2
733 {

```

We don't want a warning for redefinition of the column type. That's why we use `\@@_newcolumnntype` and not `\newcolumnntype`.

```

734   \@@_newcolumnntype #1 [ 1 ]
735   {
736     > {
737       \@@_Cell:
738       \begin { minipage } [ #2 ] { ##1 }
739       \mode_leave_vertical: \box_use:N \@arstrutbox
740     }

```

Here, we put `c` but we would have the result with `l` or `r`.

```

741       c
742       < { \box_use:N \@arstrutbox \end { minipage } \@@_end_Cell: }
743     }
744   }

```

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

```

745 \cs_new_protected:Npn \@@_pre_array:
746 {

```

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 ³¹.

```

747   \bool_if:NT \c_@@_booktabs_loaded_bool
748   { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
749   \box_clear_new:N \l_@@_cell_box
750   \cs_if_exist:NT \theiRow
751   { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
752   \int_gzero_new:N \c@iRow
753   \cs_if_exist:NT \thejCol
754   { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
755   \int_gzero_new:N \c@jCol
756   \normalbaselines

```

³¹cf. `\nicematrix@redefine@check@rerun`

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}`).

```

757 \bool_if:NT \l_@@_small_bool
758 {
759     \cs_set:Npn \arraystretch { 0.47 }
760     \dim_set:Nn \arraycolsep { 1.45 pt }
761 }

```

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`.

```

762 \cs_set:Npn \ialign
763 {
764     \bool_if:NT \l_@@_NiceTabular_bool
765     { \dim_set_eq:NN \arraycolsep \@@_old_arraycolsep_dim }
766     \bool_if:NTF \c_@@_colortbl_loaded_bool
767     {
768         \CT@everycr
769         {
770             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
771             \@@_everycr:
772         }
773     }
774     { \everycr { \@@_everycr: } }
775     \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 values of `\arraystretch`³² 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`.

```

776 \dim_gzero_new:N \g_@@_dp_row_zero_dim
777 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
778 \dim_gzero_new:N \g_@@_ht_row_zero_dim
779 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
780 \dim_gzero_new:N \g_@@_ht_row_one_dim
781 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
782 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
783 \dim_gzero_new:N \g_@@_ht_last_row_dim
784 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
785 \dim_gzero_new:N \g_@@_dp_last_row_dim
786 \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`.³³

```

787 \cs_set_eq:NN \ialign \@@_old_ialign:
788 \halign
789 }

```

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).

```

790 \cs_set_eq:NN \@@_old_ldots \ldots
791 \cs_set_eq:NN \@@_old_cdots \cdots
792 \cs_set_eq:NN \@@_old_vdots \vdots
793 \cs_set_eq:NN \@@_old_ddots \ddots

```

³²The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

³³The user will probably not use directly `\ialign` in the array... but more likely environments that utilize `\ialign` internally (e.g.: `{substack}`).

```

794 \cs_set_eq:NN \l_@@_old_iddots \iddots
795 \cs_set_eq:NN \firsthline \hline
796 \cs_set_eq:NN \lasthline \hline
797 \bool_if:NTF \l_@@_standard_cline_bool
798 { \cs_set_eq:NN \cline \@@_standard_cline }
799 { \cs_set_eq:NN \cline \@@_cline }
800 \cs_set_eq:NN \Ldots \@@_Ldots
801 \cs_set_eq:NN \Cdots \@@_Cdots
802 \cs_set_eq:NN \Vdots \@@_Vdots
803 \cs_set_eq:NN \Ddots \@@_Ddots
804 \cs_set_eq:NN \Iddots \@@_Iddots
805 \cs_set_eq:NN \hdottedline \@@_hdottedline:
806 \cs_set_eq:NN \Hspace \@@_Hspace:
807 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
808 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
809 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
810 \cs_set_eq:NN \Block \@@_Block:
811 \cs_set_eq:NN \rotate \@@_rotate:
812 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
813 \cs_set_eq:NN \dotfill \@@_dotfill:
814 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:n
815 \cs_set_eq:NN \diagbox \@@_diagbox:nn
816 \bool_if:NT \l_@@_renew_dots_bool
817 {
818   \cs_set_eq:NN \ldots \@@_Ldots
819   \cs_set_eq:NN \cdots \@@_Cdots
820   \cs_set_eq:NN \vdots \@@_Vdots
821   \cs_set_eq:NN \ddots \@@_Ddots
822   \cs_set_eq:NN \iddots \@@_Iddots
823   \cs_set_eq:NN \dots \@@_Ldots
824   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
825 }

```

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).

```

826 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
827 \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).

```

828 \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`).

```

829 \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.

```

830 \int_gzero_new:N \g_@@_col_total_int
831 \cs_set_eq:NN \@ifnextchar \new@ifnextchar

```

We define the new column types L, C and R that must be used instead of l, c and r in the preamble of `{NiceArray}`. We use `\@@_newcolumnmtype` because it will be slightly quicker than `\newcolumnmtype`.

```

832 \@@_newcolumnmtype L { > \@@_Cell: l < \@@_end_Cell: }
833 \@@_newcolumnmtype C { > \@@_Cell: c < \@@_end_Cell: }
834 \@@_newcolumnmtype R { > \@@_Cell: r < \@@_end_Cell: }

```

We redefine the column types `p`, `m` and `b`. The command `\@@_define_columntype:nn` is only used here.

```

835 \@@_define_columntype:nn p t
836 \@@_define_columntype:nn m c
837 \@@_define_columntype:nn b b

```

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.

```

838 \@@_newcolumntype w [ 2 ]
839 {
840   > {
841     \hbox_set:Nw \l_@@_cell_box
842     \@@_Cell:
843   }
844   c
845   < {
846     \@@_end_Cell:
847     \hbox_set_end:

```

The `\str_lowercase:n` is only for giving the user the ability to write `wC{1cm}` instead of `wc{1cm}` for homogeneity with the letters `L`, `C` and `R` used elsewhere in the preamble instead of `l`, `c` and `r`.

```

848     \makebox [ ##2 ] [ \str_lowercase:n { ##1 } ]
849     { \box_use_drop:N \l_@@_cell_box }
850   }
851 }
852 \@@_newcolumntype W [ 2 ]
853 {
854   > {
855     \hbox_set:Nw \l_@@_cell_box
856     \@@_Cell:
857   }
858   c
859   < {
860     \@@_end_Cell:
861     \hbox_set_end:
862     \cs_set_eq:NN \hss \hfil
863     \makebox [ ##2 ] [ \str_lowercase:n { ##1 } ]
864     { \box_use_drop:N \l_@@_cell_box }
865   }
866 }

```

By default, the letter used to specify a dotted line in the preamble of an environment of `nicematrix` (for example in `{pNiceArray}`) is the letter `.`. However, this letter is used by some packages, for example `arydshln`. That's why it's possible to change the letter used by `nicematrix` with the option `letter-for-dotted-lines` which changes the value of `\l_@@_letter_for_dotted_lines_str`. We rescan this string (which is always of length 1) in particular for the case where `pdflatex` is used with `french-babel` (the colon is activated by `french-babel` at the beginning of the document).

```

867 \tl_set_rescan:Nno
868 \l_@@_letter_for_dotted_lines_str { } \l_@@_letter_for_dotted_lines_str
869 \exp_args:NV \newcolumntype \l_@@_letter_for_dotted_lines_str
870 {
871   !
872   {

```

The following code 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).

```

873 \int_compare:nNnF \c@iRow = 0
874 {
875   \int_compare:nNnF \c@iRow = \l_@@_last_row_int
876   { \skip_horizontal:N 2\l_@@_radius_dim }
877 }

```

Consider the following code:

```
\begin{NiceArray}{C:CC:C}
a & b
c & d \\\
e & f & g & h \\\
i & j & k & l
\end{NiceArray}
```

The first “:” in the preamble will be encountered during the first row of the environment `{NiceArray}` but the second one will be encountered only in the third row. We have to issue a command `\vdottedline:n` in the code-after only one time for each “:” in the preamble. That’s why we keep a counter `\g_@@_last_vdotted_col_int` and with this counter, we know whether a letter “:” encountered during the parsing has already been taken into account in the code-after.

```
878 \int_compare:nNt \c@jCol > \g_@@_last_vdotted_col_int
879 {
880 \int_gset_eq:NN \g_@@_last_vdotted_col_int \c@jCol
881 \tl_gput_right:Nx \g_@@_internal_code_after_tl
```

The command `\@@_vdottedline:n` is protected, and, therefore, won’t be expanded before writing on `\g_@@_internal_code_after_tl`.

```
882 { \@@_vdottedline:n { \int_use:N \c@jCol } }
883 }
884 }
885 }
886 \int_gzero_new:N \g_@@_last_vdotted_col_int
887 \bool_if:NT \c_@@_siunitx_loaded_bool \@@_renew_NC@rewrite@S:
888 \int_gset:Nn \g_@@_last_vdotted_col_int { -1 }
889 \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.

```
890 \tl_gclear_new:N \g_@@_Cdots_lines_tl
891 \tl_gclear_new:N \g_@@_Ldots_lines_tl
892 \tl_gclear_new:N \g_@@_Vdots_lines_tl
893 \tl_gclear_new:N \g_@@_Ddots_lines_tl
894 \tl_gclear_new:N \g_@@_Iddots_lines_tl
895 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl
896 }
```

The environment `{NiceArrayWithDelims}`

```
897 \NewDocumentEnvironment { NiceArrayWithDelims } { m m O { } m ! O { } }
898 {
```

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.

```
899 \bgroup
900 \tl_set:Nn \l_@@_left_delim_tl { #1 }
901 \tl_set:Nn \l_@@_right_delim_tl { #2 }
902 \bool_gset_false:N \g_@@_row_of_col_done_bool
903 \str_if_empty:NT \g_@@_name_env_str
904 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
905 \@@_adapt_S_column:
906 \bool_if:NTF \l_@@_NiceTabular_bool
907 \mode_leave_vertical:
908 \@@_test_if_math_mode:
909 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
910 \bool_set_true:N \l_@@_in_env_bool
```

The command `\CT@arc@` contains the instruction of color for the rules of the array³⁴. 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.

```
911 \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).

```
912 \cs_if_exist:NT \tikz@library@external@loaded
913 {
914   \tikzset { external / export = false }
915   \cs_if_exist:NT \ifstandalone
916     { \tikzset { external / optimize = false } }
917 }
```

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

```
918 \int_gincr:N \g_@@_env_int
919 \bool_if:NF \l_@@_block_auto_columns_width_bool
920 { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

We do a redefinition of `\@arrayrule` because we want that the vertical rules drawn by `|` in the preamble of the array don't extend in the potential exterior rows.

```
921 \cs_set_protected:Npn \@arrayrule { \@addtopreamble \@@_vline: }
```

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.

```
922 \seq_clear:N \g_@@_blocks_seq
923 \seq_clear:N \g_@@_pos_of_blocks_seq
```

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`.

```
924 \bool_if:NTF \l_@@_NiceArray_bool
925 { \keys_set:nn { NiceMatrix / NiceArray } }
926 { \keys_set:nn { NiceMatrix / pNiceArray } }
927 { #3 , #5 }

928 \tl_if_empty:NF \l_@@_rules_color_tl
929 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
```

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.

```
930 \bool_if:NT \l_@@_code_before_bool
931 {
932   \seq_if_exist:cT { @@_size_ \int_use:N \g_@@_env_int _ seq }
933   {
```

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).

```
934   \int_zero_new:N \c@iRow
935   \int_set:Nn \c@iRow
936     { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
937   \int_zero_new:N \c@jCol
938   \int_set:Nn \c@jCol
939     { \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.

```
940 \int_compare:nNnF \l_@@_last_row_int = { -2 }
```

³⁴e.g. `\color[rgb]{0.5,0.5,0}`

```

941 { \int_decr:N \c@iRow }
942 \int_compare:nNnF \l_@@_last_col_int = { -2 }
943 { \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.

```

944 \pgfsys@markposition { \@@_env: - position }
945 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
946 \pgfpicture

```

First, the creation of the row nodes.

```

947 \int_step_inline:nnn
948 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 1 }
949 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 + 1 }
950 {
951   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
952   \pgfcoordinate { \@@_env: - row - ##1 }
953   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
954 }

```

Now, the creation of the col nodes.

```

955 \int_step_inline:nnn
956 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
957 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 + 1 }
958 {
959   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
960   \pgfcoordinate { \@@_env: - col - ##1 }
961   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
962 }
963 \endpgfpicture
964 \group_begin:
965   \bool_if:NT \c_@@_tikz_loaded_bool
966   {
967     \tikzset
968     {
969       every-picture / .style =
970       { overlay , name~prefix = \@@_env: - }
971     }
972   }
973   \cs_set_eq:NN \cellcolor \@@_cellcolor
974   \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
975   \cs_set_eq:NN \rowcolor \@@_rowcolor
976   \cs_set_eq:NN \rowcolors \@@_rowcolors
977   \cs_set_eq:NN \columncolor \@@_columncolor
978   \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.

```

979   \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
980   \l_@@_code_before_tl
981   \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
982 \group_end:
983 }
984 }

```

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).

```

985 \int_compare:nNnT \l_@@_last_row_int > { -2 }
986 {
987   \tl_put_right:Nn \@@_update_for_first_and_last_row:
988   {
989     \dim_gset:Nn \g_@@_ht_last_row_dim
990     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
991     \dim_gset:Nn \g_@@_dp_last_row_dim
992     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }

```

```

993     }
994   }
995   \int_compare:nNnT \l_@@_last_row_int = { -1 }
996   {
997     \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.

```

998   \str_if_empty:NTF \l_@@_name_str
999   {
1000     \cs_if_exist:cT { @@_last_row_ \int_use:N \g_@@_env_int }
1001     {
1002       \int_set:Nn \l_@@_last_row_int
1003       { \use:c { @@_last_row_ \int_use:N \g_@@_env_int } }
1004     }
1005   }
1006   {
1007     \cs_if_exist:cT { @@_last_row_ \l_@@_name_str }
1008     {
1009       \int_set:Nn \l_@@_last_row_int
1010       { \use:c { @@_last_row_ \l_@@_name_str } }
1011     }
1012   }
1013 }

```

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).

```

1014   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1015   {
1016     \str_if_empty:NTF \l_@@_name_str
1017     {
1018       \cs_if_exist:cT { @@_last_col_ \int_use:N \g_@@_env_int }
1019       {
1020         \int_set:Nn \l_@@_last_col_int
1021         { \use:c { @@_last_col_ \int_use:N \g_@@_env_int } }
1022       }
1023     }
1024     {
1025       \cs_if_exist:cT { @@_last_col_ \l_@@_name_str }
1026       {
1027         \int_set:Nn \l_@@_last_col_int
1028         { \use:c { @@_last_col_ \l_@@_name_str } }
1029       }
1030     }
1031   }

```

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

```

1032   \@@_pre_array:

```

We compute the width of the two delimiters.

```

1033   \dim_zero_new:N \l_@@_left_delim_dim
1034   \dim_zero_new:N \l_@@_right_delim_dim
1035   \bool_if:NTF \l_@@_NiceArray_bool
1036   {
1037     \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1038     \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1039   }
1040   {

```

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

```

1041   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 #1 $ }
1042   \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1043   \hbox_set:Nn \l_tmpa_box { $\bBigg@ 5 #2 $ }
1044   \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }

```

```
1045 }
```

The array will be composed in a box (named `\l_@@_the_array_box`) because we have to do manipulations concerning the potential exterior rows.

```
1046 \box_clear_new:N \l_@@_the_array_box
```

We construct the preamble of the array in `\l_tmpa_tl`.

```
1047 \tl_set:Nn \l_tmpa_tl { #4 }
1048 \int_compare:nNnTF \l_@@_first_col_int = 0
1049 { \tl_put_left:NV \l_tmpa_tl \c_@@_preamble_first_col_tl }
1050 {
1051   \bool_lazy_all:nT
1052   {
1053     \l_@@_NiceArray_bool
1054     { \bool_not_p:n \l_@@_NiceTabular_bool }
1055     { \bool_not_p:n \l_@@_vlines_bool }
1056     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1057   }
1058   { \tl_put_left:Nn \l_tmpa_tl { @ { } } }
1059 }
1060 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1061 { \tl_put_right:NV \l_tmpa_tl \c_@@_preamble_last_col_tl }
1062 {
1063   \bool_lazy_all:nT
1064   {
1065     \l_@@_NiceArray_bool
1066     { \bool_not_p:n \l_@@_NiceTabular_bool }
1067     { \bool_not_p:n \l_@@_vlines_bool }
1068     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1069   }
1070   { \tl_put_right:Nn \l_tmpa_tl { @ { } } }
1071 }
1072 \tl_put_right:Nn \l_tmpa_tl { > { \@@_error_too_much_cols: } L }
```

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).

```
1073 \hbox_set:Nw \l_@@_the_array_box
```

Here is a trick. We will call `\array` and, at the beginning, `\array` will set `\col@sep` equal to the current value of `\arraycolsep`. In we are in an environment `{NiceTabular}`, we would like that `\array` sets `\col@sep` equal to the current value of `\tabcolsep`. That's why we set `\arraycolsep` equal to `\tabcolsep`. However, the value of `\tabcolsep` in each cell of the array should be equal to the current value of `\tabcolsep` outside `{NiceTabular}`. That's why we save the current value of `\arraycolsep` and we will restore the value just before the `\halign`. It's possible because we do a redefinition of `\ialign` (see just below).

```
1074 \bool_if:NT \l_@@_NiceTabular_bool
1075 {
1076   \dim_set_eq:NN \@@_old_arraycolsep_dim \arraycolsep
1077   \dim_set_eq:NN \arraycolsep \tabcolsep
1078 }
```

If the key `\vlines` is used, we increase `\arraycolsep` by `0.5\arrayrulewidth` in order to reserve space for the width of the vertical rules drawn with Tikz after the end of the array. However, the first `\arraycolsep` is used once (between columns, `\arraycolsep` is used twice). That's why we add a `0.5\arrayrulewidth` more.

```
1079 \bool_if:NT \l_@@_vlines_bool
1080 {
1081   \dim_add:Nn \arraycolsep { 0.5 \arrayrulewidth }
1082   \skip_horizontal:N 0.5\arrayrulewidth
1083 }
1084 \skip_horizontal:N \l_@@_left_margin_dim
1085 \skip_horizontal:N \l_@@_extra_left_margin_dim
1086 \c_math_toggle_token
```

```

1087 \bool_if:NTF \l_@@_light_syntax_bool
1088 { \use:c { @@-light-syntax } }
1089 { \use:c { @@-normal-syntax } }
1090 }
1091 {
1092 \bool_if:NTF \l_@@_light_syntax_bool
1093 { \use:c { end @@-light-syntax } }
1094 { \use:c { end @@-normal-syntax } }
1095 \c_math_toggle_token
1096 \skip_horizontal:N \l_@@_right_margin_dim
1097 \skip_horizontal:N \l_@@_extra_right_margin_dim

```

If the key `\vlines` is used, we have increased `\arraycolsep` by `0.5\arrayrulewidth` in order to reserve space for the width of the vertical rules drawn with Tikz after the end of the array. However, the last `\arraycolsep` is used once (between columns, `\arraycolsep` is used twice). That's we add a `0.5 \arrayrulewidth` more.

```

1098 \bool_if:NT \l_@@_vlines_bool { \skip_horizontal:N 0.5\arrayrulewidth }
1099 \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).

```

1100 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1101 {
1102 \bool_if:NF \l_@@_last_row_without_value_bool
1103 {
1104 \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1105 {
1106 \@@_error:n { Wrong~last~row }
1107 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1108 }
1109 }
1110 }

```

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”.³⁵

```

1111 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1112 \bool_if:nT \g_@@_last_col_found_bool { \int_gdecr:N \c@jCol }

```

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

```

1113 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1114 \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. 72).

```

1115 \int_compare:nNnT \l_@@_first_col_int = 0
1116 {
1117 \skip_horizontal:N \arraycolsep
1118 \skip_horizontal:N \g_@@_width_first_col_dim
1119 }

```

The construction of the real box is different in `{NiceArray}` and in the other environments because, in `{NiceArray}`, we have to take into account the value of `baseline` and we have no delimiter to put. We begin with `{NiceArray}`.

```

1120 \bool_if:NTF \l_@@_NiceArray_bool
1121 {

```

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`.

```

1122 \str_if_eq:VnTF \l_@@_baseline_str { b }
1123 {

```

³⁵We remind that the potential “first column” (exterior) has the number 0.

```

1124 \pgfpicture
1125 \@@_qpoint:n { row - 1 }
1126 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1127 \@@_qpoint:n { row - \int_use:N \c@iRow - base }
1128 \dim_gsub:Nn \g_tmpa_dim \pgf@y
1129 \endpgfpicture
1130 \int_compare:nNt \l_@@_first_row_int = 0
1131 {
1132   \dim_gadd:Nn \g_tmpa_dim
1133     { \g_@@_ht_row_zero_dim + \g_@@_dp_row_zero_dim }
1134 }
1135 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_@@_the_array_box }
1136 }
1137 {
1138   \str_if_eq:VnTF \l_@@_baseline_str { c }
1139   { \box_use_drop:N \l_@@_the_array_box }
1140   {

```

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

```

1141 \str_if_eq:VnTF \l_@@_baseline_str { t }
1142 { \str_set:Nn \l_@@_baseline_str { 1 } }

```

Now, we convert the value of `\l_@@_baseline_str` (which should represent an integer) to an integer stored in `\l_tmpa_int`.

```

1143 \int_set:Nn \l_tmpa_int \l_@@_baseline_str
1144 \bool_if:nT
1145 {
1146   \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int
1147   || \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int
1148 }
1149 {
1150   \@@_error:n { bad-value-for-baseline }
1151   \int_set:Nn \l_tmpa_int 1
1152 }
1153 \pgfpicture
1154 \@@_qpoint:n { row - 1 }
1155 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1156 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
1157 \dim_gsub:Nn \g_tmpa_dim \pgf@y
1158 \endpgfpicture
1159 \int_compare:nNt \l_@@_first_row_int = 0
1160 {
1161   \dim_gadd:Nn \g_tmpa_dim
1162     { \g_@@_ht_row_zero_dim + \g_@@_dp_row_zero_dim }
1163 }
1164 \box_move_up:nn \g_tmpa_dim
1165 { \box_use_drop:N \l_@@_the_array_box }
1166 }
1167 }
1168 }

```

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).

```

1169 {
1170   \int_compare:nNtTF \l_@@_first_row_int = 0
1171   {
1172     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1173     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1174   }
1175   { \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”.³⁶

³⁶A 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).

```

1176 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1177 {
1178     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1179     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1180 }
1181 { \dim_zero:N \l_tmpb_dim }
1182 \hbox_set:Nn \l_tmpa_box
1183 {
1184     \c_math_toggle_token
1185     \left #1
1186     \vcenter
1187     {

```

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.

```

1188         \skip_vertical:N -\l_tmpa_dim
1189         \hbox
1190         {
1191             \bool_if:NTF \l_@@_NiceTabular_bool
1192             { \skip_horizontal:N -\tabcolsep }
1193             { \skip_horizontal:N -\arraycolsep }
1194             \box_use_drop:N \l_@@_the_array_box
1195             \bool_if:NTF \l_@@_NiceTabular_bool
1196             { \skip_horizontal:N -\tabcolsep }
1197             { \skip_horizontal:N -\arraycolsep }
1198         }

```

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

```

1199         \skip_vertical:N -\l_tmpb_dim
1200     }
1201     \right #2
1202     \c_math_toggle_token
1203 }

```

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.

```

1204 \bool_if:NTF \l_@@_max_delimiter_width_bool
1205 { \@@_put_box_in_flow_bis:nn { #1 } { #2 } }
1206 \@@_put_box_in_flow:
1207 }

```

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. 72).

```

1208 \bool_if:NT \g_@@_last_col_found_bool
1209 {
1210     \skip_horizontal:N \g_@@_width_last_col_dim
1211     \skip_horizontal:N \arraycolsep
1212 }
1213 \@@_after_array:
1214 \egroup
1215 }

```

This is the end of the environment `{NiceArrayWithDelims}`.

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`).

```

1216 \cs_new_protected:Npn \@@_put_box_in_flow:
1217 {
1218     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
1219     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
1220     \str_if_eq:VnTF \l_@@_baseline_str { c }
1221     { \box_use_drop:N \l_tmpa_box }

```

```

1222 \@@_put_box_in_flow_i:
1223 }

```

The command `\@@_put_box_in_flow_i:` is used when the value of `\l_@@_baseline_str` is different of `c` (which is the initial value and the most used).

```

1224 \cs_new_protected:Npn \@@_put_box_in_flow_i:
1225 {
1226   \str_case:NnF \l_@@_baseline_str
1227   {
1228     { t } { \int_set:Nn \l_tmpa_int 1 }
1229     { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
1230   }
1231   { \int_set:Nn \l_tmpa_int \l_@@_baseline_str }
1232   \bool_if:nT
1233   {
1234     \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int
1235     || \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int
1236   }
1237   {
1238     \@@_error:n { bad~value~for~baseline }
1239     \int_set:Nn \l_tmpa_int 1
1240   }
1241   \pgfpicture
1242     \@@_qpoint:n { row - 1 }
1243     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1244     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
1245     \dim_gadd:Nn \g_tmpa_dim \pgf@y
1246     \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).

```

1247     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
1248     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

We take into account the position of the mathematical axis.

```

1249     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }

```

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

```

1250   \endpgfpicture
1251   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
1252   \box_use_drop:N \l_tmpa_box
1253 }

```

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.

```

1254 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
1255 {

```

We will compute the real width of both delimiters used.

```

1256   \dim_zero_new:N \l_@@_real_left_delim_dim
1257   \dim_zero_new:N \l_@@_real_right_delim_dim
1258   \hbox_set:Nn \l_tmpb_box
1259   {
1260     \c_math_toggle_token
1261     \left #1
1262     \vcenter
1263     {
1264       \vbox_to_ht:nn
1265       { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
1266       { }
1267     }
1268     \right .
1269     \c_math_toggle_token
1270   }
1271   \dim_set:Nn \l_@@_real_left_delim_dim

```

```

1272     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
1273     \hbox_set:Nn \l_tmpb_box
1274     {
1275         \c_math_toggle_token
1276         \left .
1277         \vbox_to_ht:nn
1278         { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
1279         { }
1280         \right #2
1281         \c_math_toggle_token
1282     }
1283     \dim_set:Nn \l_@@_real_right_delim_dim
1284     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

1285     \skip_horizontal:N \l_@@_left_delim_dim
1286     \skip_horizontal:N -\l_@@_real_left_delim_dim
1287     \@@_put_box_in_flow:
1288     \skip_horizontal:N \l_@@_right_delim_dim
1289     \skip_horizontal:N -\l_@@_real_right_delim_dim
1290 }

```

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 used 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).

```

1291 \NewDocumentEnvironment { @@-normal-syntax } { }

```

First, we test whether the environment is empty. If it is empty, we raise a fatal error (it's only a security). In order to detect whether it is empty, we test whether the next token is `\end` and, if it's the case, we test if this is the end of the environment (if it is not, an standard error will be raised by LaTeX for incorrect nested environments).

```

1292 {
1293     \peek_meaning_ignore_spaces:NTF \end
1294     { \@@_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`).

```

1295     { \exp_args:NV \@@_array: \l_tmpa_tl }
1296 }
1297 {
1298     \@@_create_col_nodes:
1299     \endarray
1300 }

```

When the key `light-syntax` is used, we use an environment which takes its whole body as an argument (with the specifier `b` of `xparse`).

```

1301 \NewDocumentEnvironment { @@-light-syntax } { b }
1302 {

```

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`.

```

1303     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
1304     \tl_map_inline:nn { #1 }
1305     {
1306         \tl_if_eq:nnT { ##1 } { & }
1307         { \@@_fatal:n { ampersand-in-light-syntax } }
1308         \tl_if_eq:nnT { ##1 } { \ }
1309         { \@@_fatal:n { double-backslash-in-light-syntax } }
1310     }

```

Now, you extract the `code-after` or 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_@@_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to *no-op* before the execution of `\g_@@_code_after_tl`.

```
1311 \@@_light_syntax_i #1 \CodeAfter \q_stop
1312 }
```

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`.

```
1313 { }
1314 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
1315 {
1316 \tl_gput_right:Nn \g_@@_code_after_tl { #2 }
```

The body of the array, which is stored in the argument #1, is now splitted into items (and *not* tokens).

```
1317 \seq_gclear_new:N \g_@@_rows_seq
1318 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
1319 \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.

```
1320 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1321 { \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`).

```
1322 \exp_args:NV \@@_array: \l_tmpa_tl
```

We need a global affectation because, when executing `\l_tmpa_tl`, we will exit the first cell of the array.

```
1323 \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
1324 \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
1325 \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
1326 \@@_create_col_nodes:
1327 \endarray
1328 }
1329 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
1330 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
1331 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
1332 {
1333 \seq_gclear_new:N \g_@@_cells_seq
1334 \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
1335 \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
1336 \l_tmpa_tl
1337 \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
1338 }
```

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).

```
1339 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
1340 {
1341 \str_if_eq:VnT \g_@@_name_env_str { #2 }
1342 { \@@_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.

```
1343 \end { #2 }
1344 }
```

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).

```
1345 \cs_new:Npn \@@_create_col_nodes:
1346 {
```

```

1347 \crrc
1348 \int_compare:nNnT \c@iRow = 0 { \@@_fatal:n { Zero~row } }
1349 \int_compare:nNnT \l_@@_first_col_int = 0
1350 {
1351   \omit
1352   \skip_horizontal:N -2\col@sep
1353   \bool_if:NT \l_@@_code_before_bool
1354     { \pgfsys@markposition { \@@_env: - col - 0 } }
1355   \pgfpicture
1356   \pgfrememberpicturepositiononpagetrue
1357   \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
1358   \str_if_empty:NF \l_@@_name_str
1359     { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
1360   \endpgfpicture
1361   &
1362 }
1363 \omit

```

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

```

1364 \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`).

```

1365 \int_compare:nNnTF \l_@@_first_col_int = 0
1366 {
1367   \bool_if:NT \l_@@_code_before_bool
1368   {
1369     \hbox
1370     {
1371       \skip_horizontal:N -0.5\arrayrulewidth
1372       \pgfsys@markposition { \@@_env: - col - 1 }
1373       \skip_horizontal:N 0.5\arrayrulewidth
1374     }
1375   }
1376   \pgfpicture
1377   \pgfrememberpicturepositiononpagetrue
1378   \pgfcoordinate { \@@_env: - col - 1 }
1379     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
1380   \str_if_empty:NF \l_@@_name_str
1381     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
1382   \endpgfpicture
1383 }
1384 {
1385   \bool_if:NT \l_@@_code_before_bool
1386   {
1387     \hbox
1388     {
1389       \skip_horizontal:N 0.5 \arrayrulewidth
1390       \pgfsys@markposition { \@@_env: - col - 1 }
1391       \skip_horizontal:N -0.5\arrayrulewidth
1392     }
1393   }
1394   \pgfpicture
1395   \pgfrememberpicturepositiononpagetrue
1396   \pgfcoordinate { \@@_env: - col - 1 }
1397     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
1398   \str_if_empty:NF \l_@@_name_str
1399     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
1400   \endpgfpicture
1401 }

```

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 erased by a fixed value in the concerned cases.

```

1402 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
1403 \bool_if:NF \l_@@_auto_columns_width_bool
1404 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
1405 {
1406   \bool_lazy_and:nnTF
1407     \l_@@_auto_columns_width_bool
1408     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
1409     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
1410     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
1411     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
1412 }
1413 \skip_horizontal:N \g_tmpa_skip
1414 \hbox
1415 {
1416   \bool_if:NT \l_@@_code_before_bool
1417   {
1418     \hbox
1419     {
1420       \skip_horizontal:N -0.5\arrayrulewidth
1421       \pgfsys@markposition { \@@_env: - col - 2 }
1422       \skip_horizontal:N 0.5\arrayrulewidth
1423     }
1424   }
1425   \pgfpicture
1426   \pgfrememberpicturepositiononpagetrue
1427   \pgfcoordinate { \@@_env: - col - 2 }
1428   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
1429   \str_if_empty:NF \l_@@_name_str
1430   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
1431   \endpgfpicture
1432 }

```

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.

```

1433 \int_gset:Nn \g_tmpa_int 1
1434 \bool_if:NNTF \g_@@_last_col_found_bool
1435 { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
1436 { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
1437 {
1438   &
1439   \omit

```

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

```

1440 \int_gincr:N \g_tmpa_int
1441 \skip_horizontal:N \g_tmpa_skip
1442 \bool_if:NT \l_@@_code_before_bool
1443 {
1444   \hbox
1445   {
1446     \skip_horizontal:N -0.5\arrayrulewidth
1447     \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1448     \skip_horizontal:N 0.5\arrayrulewidth
1449   }
1450 }

```

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

```

1451 \pgfpicture
1452 \pgfrememberpicturepositiononpagetrue
1453 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1454 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
1455 \str_if_empty:NF \l_@@_name_str
1456 {
1457   \pgfnodealias

```

```

1458         { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
1459         { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1460     }
1461     \endpgfpicture
1462 }
1463 \bool_if:NT \g_@@_last_col_found_bool
1464 {
1465     \bool_if:NT \l_@@_code_before_bool
1466     {
1467         \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1468     }
1469     \skip_horizontal:N 2\col@sep
1470     \pgfpicture
1471     \pgfrememberpicturepositiononpagetrue
1472     \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1473     \pgfpointorigin
1474     \str_if_empty:NF \l_@@_name_str
1475     {
1476         \pgfnodealias
1477         { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
1478         { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1479     }
1480     \endpgfpicture
1481     \skip_horizontal:N -2\col@sep
1482 }
1483 \cr
1484 }

```

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

```

1485 \tl_const:Nn \c_@@_preamble_first_col_tl
1486 {
1487     >
1488     {
1489         \@@_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.

```

1490         \hbox_set:Nw \l_@@_cell_box
1491         \@@_math_toggle_token:
1492         \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”.

```

1493         \bool_lazy_and:nnT
1494         { \int_compare_p:nNn \c@iRow > 0 }
1495         {
1496             \bool_lazy_or_p:nn
1497             { \int_compare_p:nNn \l_@@_last_row_int < 0 }
1498             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
1499         }
1500         {
1501             \l_@@_code_for_first_col_tl
1502             \xglobal \colorlet { nicematrix-first-col } { . }
1503         }
1504     }

```

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`.

```

1505         l
1506         <
1507         {
1508             \@@_math_toggle_token:
1509             \hbox_set_end:
1510             \@@_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.

```

1511 \dim_gset:Nn \g_@@_width_first_col_dim
1512 { \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.

```

1513 \hbox_overlap_left:n
1514 {
1515   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1516     \@@_node_for_the_cell:
1517     { \box_use_drop:N \l_@@_cell_box }
1518     \skip_horizontal:N \l_@@_left_delim_dim
1519     \skip_horizontal:N \l_@@_left_margin_dim
1520     \skip_horizontal:N \l_@@_extra_left_margin_dim
1521   }
1522   \skip_horizontal:N -2\col@sep
1523 }
1524 }

```

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

```

1525 \tl_const:Nn \c_@@_preamble_last_col_tl
1526 {
1527   >
1528   {

```

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

```

1529 \bool_gset_true:N \g_@@_last_col_found_bool
1530 \int_gincr:N \c@jCol
1531 \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.

```

1532 \hbox_set:Nw \l_@@_cell_box
1533 \@@_math_toggle_token:
1534 \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”.

```

1535 \int_compare:nNnT \c@iRow > 0
1536 {
1537   \bool_lazy_or:nnT
1538   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
1539   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
1540   {
1541     \l_@@_code_for_last_col_tl
1542     \xglobal \colorlet { nicematrix-last-col } { . }
1543   }
1544 }
1545 }
1546 1
1547 <
1548 {
1549   \@@_math_toggle_token:
1550   \hbox_set_end:
1551   \@@_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.

```

1552 \dim_gset:Nn \g_@@_width_last_col_dim
1553 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
1554 \skip_horizontal:N -2\col@sep

```

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

```

1555 \hbox_overlap_right:n
1556 {
1557   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1558   {
1559     \skip_horizontal:N \l_@@_right_delim_dim

```

```

1560         \skip_horizontal:N \l_@@_right_margin_dim
1561         \skip_horizontal:N \l_@@_extra_right_margin_dim
1562         \@@_node_for_the_cell:
1563     }
1564 }
1565 }
1566 }

```

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.

```

1567 \NewDocumentEnvironment { NiceArray } { }
1568 {
1569     \bool_set_true:N \l_@@_NiceArray_bool
1570     \str_if_empty:NT \g_@@_name_env_str
1571     { \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).

```

1572     \NiceArrayWithDelims . .
1573 }
1574 { \endNiceArrayWithDelims }

```

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

```

1575 \NewDocumentEnvironment { pNiceArray } { }
1576 {
1577     \str_if_empty:NT \g_@@_name_env_str
1578     { \str_gset:Nn \g_@@_name_env_str { pNiceArray } }
1579     \@@_test_if_math_mode:
1580     \NiceArrayWithDelims ( )
1581 }
1582 { \endNiceArrayWithDelims }

1583 \NewDocumentEnvironment { bNiceArray } { }
1584 {
1585     \str_if_empty:NT \g_@@_name_env_str
1586     { \str_gset:Nn \g_@@_name_env_str { bNiceArray } }
1587     \@@_test_if_math_mode:
1588     \NiceArrayWithDelims [ ]
1589 }
1590 { \endNiceArrayWithDelims }

1591 \NewDocumentEnvironment { BNiceArray } { }
1592 {
1593     \str_if_empty:NT \g_@@_name_env_str
1594     { \str_gset:Nn \g_@@_name_env_str { BNiceArray } }
1595     \@@_test_if_math_mode:
1596     \NiceArrayWithDelims \{ \}
1597 }
1598 { \endNiceArrayWithDelims }

1599 \NewDocumentEnvironment { vNiceArray } { }
1600 {
1601     \str_if_empty:NT \g_@@_name_env_str
1602     { \str_gset:Nn \g_@@_name_env_str { vNiceArray } }
1603     \@@_test_if_math_mode:
1604     \NiceArrayWithDelims | |
1605 }
1606 { \endNiceArrayWithDelims }

1607 \NewDocumentEnvironment { VNiceArray } { }
1608 {
1609     \str_if_empty:NT \g_@@_name_env_str
1610     { \str_gset:Nn \g_@@_name_env_str { VNiceArray } }

```

```

1611 \@@_test_if_math_mode:
1612 \NiceArrayWithDelims \l \l
1613 }
1614 { \endNiceArrayWithDelims }

```

The environment `{NiceMatrix}` and its variants

```

1615 \cs_new_protected:Npn \@@_define_env:n #1
1616 {
1617   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
1618   {
1619     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
1620     \tl_set:Nn \l_@@_type_of_col_tl C
1621     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
1622     \exp_args:Nnx \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
1623   }
1624   { \use:c { end #1 NiceArray } }
1625 }
1626 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
1627 {
1628   \use:c { #1 NiceArray }
1629   {
1630     *
1631     {
1632       \int_compare:nNnTF \l_@@_last_col_int < 0
1633       \c@MaxMatrixCols
1634       { \@@_pred:n \l_@@_last_col_int }
1635     }
1636     #2
1637   }
1638 }
1639 \@@_define_env:n { }
1640 \@@_define_env:n p
1641 \@@_define_env:n b
1642 \@@_define_env:n B
1643 \@@_define_env:n v
1644 \@@_define_env:n V

```

The environment `{NiceTabular}`

```

1645 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
1646 {
1647   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
1648   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
1649   \bool_set_true:N \l_@@_NiceTabular_bool
1650   \NiceArray { #2 }
1651 }
1652 { \endNiceArray }

```

After the construction of the array

```

1653 \cs_new_protected:Npn \@@_after_array:
1654 {
1655   \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.

```

1656 \bool_if:NT \g_@@_last_col_found_bool
1657 { \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`.

```

1658 \bool_if:NT \l_@@_last_col_without_value_bool
1659 {
1660   \dim_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int
1661   \iow_shipout:Nn \@mainaux \ExplSyntaxOn
1662   \iow_shipout:Nx \@mainaux
1663   {
1664     \cs_gset:cpn { @@_last_col_ \int_use:N \g_@@_env_int }
1665     { \int_use:N \g_@@_col_total_int }
1666   }
1667   \str_if_empty:NF \l_@@_name_str
1668   {
1669     \iow_shipout:Nx \@mainaux
1670     {
1671       \cs_gset:cpn { @@_last_col_ \l_@@_name_str }
1672       { \int_use:N \g_@@_col_total_int }
1673     }
1674   }
1675   \iow_shipout:Nn \@mainaux \ExplSyntaxOff
1676 }

```

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.

```

1677 \bool_if:NT \l_@@_last_row_without_value_bool
1678 {
1679   \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.

```

1680 \bool_if:NF \l_@@_light_syntax_bool
1681 {
1682   \iow_shipout:Nn \@mainaux \ExplSyntaxOn
1683   \iow_shipout:Nx \@mainaux
1684   {
1685     \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int }
1686     { \int_use:N \g_@@_row_total_int }
1687   }

```

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.

```

1688 \str_if_empty:NF \l_@@_name_str
1689 {
1690   \iow_shipout:Nx \@mainaux
1691   {
1692     \cs_gset:cpn { @@_last_row_ \l_@@_name_str }
1693     { \int_use:N \g_@@_row_total_int }
1694   }
1695 }
1696 \iow_shipout:Nn \@mainaux \ExplSyntaxOff
1697 }
1698 }

```

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

```

1699 \bool_if:NT \l_@@_code_before_bool
1700 {
1701   \iow_now:Nn \@mainaux \ExplSyntaxOn
1702   \iow_now:Nx \@mainaux
1703   { \seq_clear_new:c { @@_size _ \int_use:N \g_@@_env_int _ seq } }
1704   \iow_now:Nx \@mainaux
1705   {
1706     \seq_gset_from_clist:cn { @@_size _ \int_use:N \g_@@_env_int _ seq }
1707     {

```

```

1708         \int_use:N \l_@@_first_row_int ,
1709         \int_use:N \g_@@_row_total_int ,
1710         \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.

```

1711         \bool_lazy_and:nnTF
1712         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
1713         { \bool_not_p:n \g_@@_last_col_found_bool }
1714         \@@_succ:n
1715         \int_use:N
1716         \g_@@_col_total_int
1717     }
1718 }
1719 \iow_now:Nn \@mainaux \ExplSyntaxOff
1720 }

```

By default, the diagonal lines will be parallelized³⁷. 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.

```

1721     \bool_if:NT \l_@@_parallelize_diags_bool
1722     {
1723         \int_gzero_new:N \g_@@_ddots_int
1724         \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.

```

1725         \dim_gzero_new:N \g_@@_delta_x_one_dim
1726         \dim_gzero_new:N \g_@@_delta_y_one_dim
1727         \dim_gzero_new:N \g_@@_delta_x_two_dim
1728         \dim_gzero_new:N \g_@@_delta_y_two_dim
1729     }
1730     \bool_if:nTF \l_@@_medium_nodes_bool
1731     {
1732         \bool_if:NTF \l_@@_large_nodes_bool
1733         \@@_create_medium_and_large_nodes:
1734         \@@_create_medium_nodes:
1735     }
1736     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
1737     \int_zero_new:N \l_@@_initial_i_int
1738     \int_zero_new:N \l_@@_initial_j_int
1739     \int_zero_new:N \l_@@_final_i_int
1740     \int_zero_new:N \l_@@_final_j_int
1741     \bool_set_false:N \l_@@_initial_open_bool
1742     \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.

```

1743     \bool_if:NT \l_@@_small_bool
1744     {
1745         \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
1746         \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.

```

1747         \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
1748     }

```

Now, we actually draw the dotted lines.

³⁷It's possible to use the option `parallelize-diags` to disable this parallelization.

```

1749 \@@_draw_dotted_lines:
1750 \bool_if:NTF \l_@@_hvlines_bool
1751   \@@_draw_hvlines:
1752   {
1753     \bool_if:NT \l_@@_hlines_bool \@@_draw_hlines:
1754     \bool_if:NT \l_@@_vlines_bool \@@_draw_vlines:
1755     \bool_if:NT \l_@@_hvlines_except_corners_bool
1756       \@@_draw_hvlines_except_corners:
1757   }

```

We have to revert to a clean version of `\ialign` because there may be tabulars in the `\Block` instructions that will be composed now.

```

1758 \cs_set_eq:NN \ialign \@@_old_ialign:
1759 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
1760 \g_@@_internal_code_after_tl
1761 \tl_gclear:N \g_@@_internal_code_after_tl
1762 \bool_if:NT \c_@@_tikz_loaded_bool
1763   {
1764     \tikzset
1765     {
1766       every-picture / .style =
1767       {
1768         overlay ,
1769         remember-picture ,
1770         name-prefix = \@@_env: -
1771       }
1772     }
1773   }
1774 \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 one is eventually present in `\g_@@_code_after_tl`. That's why we set `\Code-after` to be *no-op* now.

```

1775 \cs_set_eq:NN \CodeAfter \prg_do_nothing:

```

And here's the code-after:

```

1776 \g_@@_code_after_tl
1777 \tl_gclear:N \g_@@_code_after_tl
1778 \group_end:
1779 \str_gclear:N \g_@@_name_env_str
1780 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array³⁸. 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`.

```

1781 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
1782 }

```

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).

```

1783 \AtBeginDocument
1784 {
1785   \cs_new_protected:Npx \@@_draw_dotted_lines:
1786   {
1787     \c_@@_pgfortikzpicture_tl
1788     \@@_draw_dotted_lines_i:
1789     \c_@@_endpgfortikzpicture_tl
1790   }
1791 }

```

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

³⁸e.g. `\color[rgb]{0.5,0.5,0}`)

```

1792 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
1793 {
1794   \pgfrememberpicturepositiononpagetrue
1795   \pgf@relevantforpicturesizefalse
1796   \g_@@_HVdotsfor_lines_tl
1797   \g_@@_Vdots_lines_tl
1798   \g_@@_Ddots_lines_tl
1799   \g_@@_Iddots_lines_tl
1800   \g_@@_Cdots_lines_tl
1801   \g_@@_Ldots_lines_tl
1802 }

1803 \cs_new_protected:Npn \@@_restore_iRow_jCol:
1804 {
1805   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
1806   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
1807 }

```

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.

```

1808 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
1809 {

```

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

```

1810   \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

1811   \int_set:Nn \l_@@_initial_i_int { #1 }
1812   \int_set:Nn \l_@@_initial_j_int { #2 }
1813   \int_set:Nn \l_@@_final_i_int { #1 }
1814   \int_set:Nn \l_@@_final_j_int { #2 }

```

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

```

1815     \bool_set_false:N \l_@@_stop_loop_bool
1816     \bool_do_until:Nn \l_@@_stop_loop_bool
1817     {
1818         \int_add:Nn \l_@@_final_i_int { #3 }
1819         \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

1820     \bool_set_false:N \l_@@_final_open_bool
1821     \int_compare:nNnTF \l_@@_final_i_int > \c@iRow
1822     {
1823         \int_compare:nNnTF { #3 } = 1
1824         { \bool_set_true:N \l_@@_final_open_bool }
1825         {
1826             \int_compare:nNnT \l_@@_final_j_int > \c@jCol
1827             { \bool_set_true:N \l_@@_final_open_bool }
1828         }
1829     }
1830     {
1831         \int_compare:nNnTF \l_@@_final_j_int < 1
1832         {
1833             \int_compare:nNnT { #4 } = { -1 }
1834             { \bool_set_true:N \l_@@_final_open_bool }
1835         }
1836         {
1837             \int_compare:nNnT \l_@@_final_j_int > \c@jCol
1838             {
1839                 \int_compare:nNnT { #4 } = 1
1840                 { \bool_set_true:N \l_@@_final_open_bool }
1841             }
1842         }
1843     }
1844     \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.

```

1845     {

```

We do a step backwards.

```

1846         \int_sub:Nn \l_@@_final_i_int { #3 }
1847         \int_sub:Nn \l_@@_final_j_int { #4 }
1848         \bool_set_true:N \l_@@_stop_loop_bool
1849     }

```

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`.

```

1850     {
1851         \cs_if_exist:cTF
1852         {
1853             @@ _ dotted _
1854             \int_use:N \l_@@_final_i_int -
1855             \int_use:N \l_@@_final_j_int
1856         }
1857         {
1858             \int_sub:Nn \l_@@_final_i_int { #3 }
1859             \int_sub:Nn \l_@@_final_j_int { #4 }
1860             \bool_set_true:N \l_@@_final_open_bool
1861             \bool_set_true:N \l_@@_stop_loop_bool
1862         }
1863     }
1864     \cs_if_exist:cTF
1865     {
1866         pgf @ sh @ ns @ \@@_env:
1867         - \int_use:N \l_@@_final_i_int

```

```

1868         - \int_use:N \l_@@_final_j_int
1869     }
1870     { \bool_set_true:N \l_@@_stop_loop_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be mark 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.

```

1871     {
1872         \cs_set:cpn
1873         {
1874             @@ _ dotted _
1875             \int_use:N \l_@@_final_i_int -
1876             \int_use:N \l_@@_final_j_int
1877         }
1878         { }
1879     }
1880 }
1881 }
1882 }

```

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

```

1883 \bool_set_false:N \l_@@_stop_loop_bool
1884 \bool_do_until:Nn \l_@@_stop_loop_bool
1885 {
1886     \int_sub:Nn \l_@@_initial_i_int { #3 }
1887     \int_sub:Nn \l_@@_initial_j_int { #4 }
1888     \bool_set_false:N \l_@@_initial_open_bool
1889     \int_compare:nNnTF \l_@@_initial_i_int < 1
1890     {
1891         \int_compare:nNnTF { #3 } = 1
1892         { \bool_set_true:N \l_@@_initial_open_bool }
1893         {
1894             \int_compare:nNnT \l_@@_initial_j_int = 0
1895             { \bool_set_true:N \l_@@_initial_open_bool }
1896         }
1897     }
1898     {
1899         \int_compare:nNnTF \l_@@_initial_j_int < 1
1900         {
1901             \int_compare:nNnT { #4 } = 1
1902             { \bool_set_true:N \l_@@_initial_open_bool }
1903         }
1904         {
1905             \int_compare:nNnT \l_@@_initial_j_int > \c@jCol
1906             {
1907                 \int_compare:nNnT { #4 } = { -1 }
1908                 { \bool_set_true:N \l_@@_initial_open_bool }
1909             }
1910         }
1911     }
1912     \bool_if:Ntf \l_@@_initial_open_bool
1913     {
1914         \int_add:Nn \l_@@_initial_i_int { #3 }
1915         \int_add:Nn \l_@@_initial_j_int { #4 }
1916         \bool_set_true:N \l_@@_stop_loop_bool
1917     }
1918     {
1919         \cs_if_exist:cTF

```

```

1920     {
1921         @@ _ dotted _
1922         \int_use:N \l_@@_initial_i_int -
1923         \int_use:N \l_@@_initial_j_int
1924     }
1925     {
1926         \int_add:Nn \l_@@_initial_i_int { #3 }
1927         \int_add:Nn \l_@@_initial_j_int { #4 }
1928         \bool_set_true:N \l_@@_initial_open_bool
1929         \bool_set_true:N \l_@@_stop_loop_bool
1930     }
1931     {
1932         \cs_if_exist:cTF
1933         {
1934             pgf @ sh @ ns @ \@@_env:
1935             - \int_use:N \l_@@_initial_i_int
1936             - \int_use:N \l_@@_initial_j_int
1937         }
1938         { \bool_set_true:N \l_@@_stop_loop_bool }
1939         {
1940             \cs_set:cpn
1941             {
1942                 @@ _ dotted _
1943                 \int_use:N \l_@@_initial_i_int -
1944                 \int_use:N \l_@@_initial_j_int
1945             }
1946             { }
1947         }
1948     }
1949 }
1950 }

```

If the key `hvlines` is used, 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.

```

1951     \bool_if:NT \l_@@_hvlines_bool
1952     {
1953         \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
1954         {
1955             { \int_use:N \l_@@_initial_i_int }
1956             { \int_use:N \l_@@_initial_j_int }
1957             { \int_use:N \l_@@_final_i_int }
1958             { \int_use:N \l_@@_final_j_int }
1959         }
1960     }
1961 }

1962 \cs_new_protected:Npn \@@_set_initial_coords:
1963 {
1964     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
1965     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
1966 }
1967 \cs_new_protected:Npn \@@_set_final_coords:
1968 {
1969     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
1970     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
1971 }
1972 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
1973 {
1974     \pgfpointanchor
1975     {
1976         \@@_env:
1977         - \int_use:N \l_@@_initial_i_int
1978         - \int_use:N \l_@@_initial_j_int
1979     }

```

```

1980     { #1 }
1981     \@@_set_initial_coords:
1982   }
1983   \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
1984   {
1985     \pgfpointanchor
1986     {
1987       \@@_env:
1988       - \int_use:N \l_@@_final_i_int
1989       - \int_use:N \l_@@_final_j_int
1990     }
1991     { #1 }
1992     \@@_set_final_coords:
1993   }

```

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.

```

1994   \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
1995   {
1996     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
1997     {
1998       \@@_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.

```

1999       \group_begin:
2000       \int_compare:nNnTF { #1 } = 0
2001       { \color { nicematrix-first-row } }
2002       {

```

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.

```

2003       \int_compare:nNnT { #1 } = \l_@@_last_row_int
2004       { \color { nicematrix-last-row } }
2005     }
2006     \keys_set:nn { NiceMatrix / xdots } { #3 }
2007     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2008     \@@_actually_draw_Ldots:
2009   \group_end:
2010   }
2011 }

```

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`.

```

2012   \cs_new_protected:Npn \@@_actually_draw_Ldots:
2013   {
2014     \bool_if:NTF \l_@@_initial_open_bool
2015     {
2016       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2017       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2018       \dim_add:Nn \l_@@_x_initial_dim

```

```

2019         { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
2020         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
2021         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2022     }
2023     { \@@_set_initial_coords_from_anchor:n { base~east } }
2024     \bool_if:NTF \l_@@_final_open_bool
2025     {
2026         \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2027         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2028         \dim_sub:Nn \l_@@_x_final_dim
2029         { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
2030         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
2031         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2032     }
2033     { \@@_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 texte. Of course, maybe we should not do that when the option `line-style` is used (?).

```

2034     \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
2035     \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
2036     \@@_draw_line:
2037 }

```

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.

```

2038 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
2039 {
2040     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2041     {
2042         \@@_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.

```

2043         \group_begin:
2044         \int_compare:nNnTF { #1 } = 0
2045         { \color { nicematrix-first-row } }
2046         {

```

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.

```

2047         \int_compare:nNnT { #1 } = \l_@@_last_row_int
2048         { \color { nicematrix-last-row } }
2049     }
2050     \keys_set:nn { NiceMatrix / xdots } { #3 }
2051     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2052     \@@_actually_draw_Cdots:
2053     \group_end:
2054 }
2055 }

```

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`.

```

2056 \cs_new_protected:Npn \@@_actually_draw_Cdots:
2057 {
2058   \bool_if:NTF \l_@@_initial_open_bool
2059   {
2060     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2061     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2062     \dim_add:Nn \l_@@_x_initial_dim
2063       { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
2064   }
2065   { \@@_set_initial_coords_from_anchor:n { mid-east } }
2066   \bool_if:NTF \l_@@_final_open_bool
2067   {
2068     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2069     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2070     \dim_sub:Nn \l_@@_x_final_dim
2071       { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
2072   }
2073   { \@@_set_final_coords_from_anchor:n { mid-west } }
2074   \bool_lazy_and:nnTF
2075     \l_@@_initial_open_bool
2076     \l_@@_final_open_bool
2077   {
2078     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2079     \dim_set_eq:NN \l_tmpa_dim \pgf@y
2080     \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
2081     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
2082     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
2083   }
2084   {
2085     \bool_if:NT \l_@@_initial_open_bool
2086       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
2087     \bool_if:NT \l_@@_final_open_bool
2088       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
2089   }
2090   \@@_draw_line:
2091 }

```

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.

```

2092 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
2093 {
2094   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2095   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2096   {
2097     \@@_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.

```

2098   \group_begin:
2099     \int_compare:nNnTF { #2 } = 0
2100     { \color { nicematrix-first-col } }
2101     {
2102       \int_compare:nNnT { #2 } = \l_@@_last_col_int
2103       { \color { nicematrix-last-col } }
2104     }
2105     \keys_set:nn { NiceMatrix / xdots } { #3 }
2106     \@@_actually_draw_Vdots:
2107   \group_end:
2108 }
2109 }

```

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.

```
2110 \cs_new_protected:Npn \@@_actually_draw_Vdots:
2111 {
```

The boolean \l_tmpa_bool indicates whether the column is of type l (L of {NiceArray}) or may be considered as if.

```
2112   \bool_set_false:N \l_tmpa_bool
2113   \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
2114   {
2115     \@@_set_initial_coords_from_anchor:n { south-west }
2116     \@@_set_final_coords_from_anchor:n { north-west }
2117     \bool_set:Nn \l_tmpa_bool
2118       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
2119   }
```

Now, we try to determine whether the column is of type c (C of {NiceArray}) or may be considered as if.

```
2120   \bool_if:NTF \l_@@_initial_open_bool
2121   {
2122     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2123     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2124   }
2125   { \@@_set_initial_coords_from_anchor:n { south } }
2126   \bool_if:NTF \l_@@_final_open_bool
2127   {
2128     \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2129     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2130   }
2131   { \@@_set_final_coords_from_anchor:n { north } }
2132   \bool_if:NTF \l_@@_initial_open_bool
2133   {
2134     \bool_if:NTF \l_@@_final_open_bool
2135     {
2136       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2137       \dim_set_eq:NN \l_tmpa_dim \pgf@x
2138       \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
2139       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
2140       \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.

```
2141     \int_compare:nNnT \l_@@_last_col_int > { -2 }
2142     {
2143       \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
2144       {
2145         \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
2146         \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
2147         \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
2148         \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
2149       }
2150     }
2151   }
2152   { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
2153 }
```

```

2154 {
2155     \bool_if:NTF \l_@@_final_open_bool
2156     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
2157     {

```

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.

```

2158         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
2159         {
2160             \dim_set:Nn \l_@@_x_initial_dim
2161             {
2162                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
2163                 \l_@@_x_initial_dim \l_@@_x_final_dim
2164             }
2165             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
2166         }
2167     }
2168 }
2169 \@@_draw_line:
2170 }

```

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.

```

2171 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
2172 {
2173     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2174     {
2175         \@@_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.

```

2176         \group_begin:
2177         \keys_set:nn { NiceMatrix / xdots } { #3 }
2178         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2179         \@@_actually_draw_Ddots:
2180         \group_end:
2181     }
2182 }

```

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`.

```

2183 \cs_new_protected:Npn \@@_actually_draw_Ddots:
2184 {
2185     \bool_if:NTF \l_@@_initial_open_bool
2186     {
2187         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2188         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2189         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2190         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

2191     }
2192     { \@@_set_initial_coords_from_anchor:n { south-east } }
2193     \bool_if:NTF \l_@@_final_open_bool
2194     {
2195         \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2196         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2197         \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2198         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2199     }
2200     { \@@_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.

```

2201     \bool_if:NT \l_@@_parallelize_diags_bool
2202     {
2203         \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).

```

2204         \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.

```

2205         {
2206             \dim_gset:Nn \g_@@_delta_x_one_dim
2207             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
2208             \dim_gset:Nn \g_@@_delta_y_one_dim
2209             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
2210         }

```

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`.

```

2211         {
2212             \dim_set:Nn \l_@@_y_final_dim
2213             {
2214                 \l_@@_y_initial_dim +
2215                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2216                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
2217             }
2218         }
2219     }
2220     \@@_draw_line:
2221 }

```

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.

```

2222 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
2223 {
2224     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2225     {
2226         \@@_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.

```

2227     \group_begin:
2228     \keys_set:nn { NiceMatrix / xdots } { #3 }
2229     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2230     \@@_actually_draw_Iddots:
2231     \group_end:
2232 }
2233 }

```

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.

```

2234 \cs_new_protected:Npn \@@_actually_draw_Iddots:
2235 {
2236   \bool_if:NTF \l_@@_initial_open_bool
2237   {
2238     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2239     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2240     \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
2241     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2242   }
2243   { \@@_set_initial_coords_from_anchor:n { south-west } }
2244   \bool_if:NTF \l_@@_final_open_bool
2245   {
2246     \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2247     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2248     \@@_qpoint:n { col - \int_use:N \l_@@_final_j_int }
2249     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2250   }
2251   { \@@_set_final_coords_from_anchor:n { north-east } }
2252   \bool_if:NT \l_@@_parallelize_diags_bool
2253   {
2254     \int_gincr:N \g_@@_iddots_int
2255     \int_compare:nNnTF \g_@@_iddots_int = 1
2256     {
2257       \dim_gset:Nn \g_@@_delta_x_two_dim
2258       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
2259       \dim_gset:Nn \g_@@_delta_y_two_dim
2260       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
2261     }
2262     {
2263       \dim_set:Nn \l_@@_y_final_dim
2264       {
2265         \l_@@_y_initial_dim +
2266         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2267         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
2268       }
2269     }
2270   }
2271   \@@_draw_line:
2272 }

```

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`

```

2273 \cs_new_protected:Npn \@@_draw_line:
2274 {
2275   \pgfrememberpicturepositiononpagetrue
2276   \pgf@relevantforpicturesizefalse
2277   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
2278     \@@_draw_standard_dotted_line:
2279     \@@_draw_non_standard_dotted_line:
2280 }

```

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.

```

2281 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
2282 {
2283   \begin { scope }
2284   \exp_args:No \@@_draw_non_standard_dotted_line:n
2285     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
2286 }

```

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.

```

2287 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
2288 {
2289   \draw
2290   [
2291     #1 ,
2292     shorten~> = \l_@@_xdots_shorten_dim ,
2293     shorten~< = \l_@@_xdots_shorten_dim ,
2294   ]
2295     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
2296     -- node [ sloped , above ]
2297       { \c_math_toggle_token \scriptstyle \l_@@_xdots_up_tl \c_math_toggle_token }
2298     node [ sloped , below ]
2299       {
2300         \c_math_toggle_token
2301         \scriptstyle \l_@@_xdots_down_tl
2302         \c_math_toggle_token
2303       }
2304     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
2305   \end { scope }
2306 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of points (which give a dotted line with real round points).

```

2307 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
2308 {

```

First, we put the labels.

```

2309   \bool_lazy_and:nnF
2310     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
2311     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
2312   {
2313     \pgfscope
2314     \pgftransformshift
2315     {
2316       \pgfpointlineattime { 0.5 }
2317       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
2318       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
2319     }

```

```

2320 \pgftransformrotate
2321 {
2322     \fp_eval:n
2323     {
2324         atand
2325         (
2326             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
2327             \l_@@_x_final_dim - \l_@@_x_initial_dim
2328         )
2329     }
2330 }
2331 \pgfnode
2332 { rectangle }
2333 { south }
2334 {
2335     \c_math_toggle_token
2336     \scriptstyle \l_@@_xdots_up_tl
2337     \c_math_toggle_token
2338 }
2339 { }
2340 { \pgfusepath { } }
2341 \pgfnode
2342 { rectangle }
2343 { north }
2344 {
2345     \c_math_toggle_token
2346     \scriptstyle \l_@@_xdots_down_tl
2347     \c_math_toggle_token
2348 }
2349 { }
2350 { \pgfusepath { } }
2351 \endpgfscope
2352 }
2353 \pgfrememberpicturepositiononpagetrue
2354 \pgf@relevantforpicturesizefalse
2355 \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.

```

2356 \dim_zero_new:N \l_@@_l_dim
2357 \dim_set:Nn \l_@@_l_dim
2358 {
2359     \fp_to_dim:n
2360     {
2361         sqrt
2362         (
2363             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
2364             +
2365             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
2366         )
2367     }
2368 }

```

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.

```

2369 \bool_lazy_or:nnF
2370 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
2371 { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
2372 \@@_draw_standard_dotted_line_i:
2373 \group_end:
2374 }
2375 \dim_const:Nn \c_@@_max_l_dim { 50 cm }

```

```

2376 \cs_new_protected:Npn \l_@@_draw_standard_dotted_line_i:
2377 {

```

The integer `\l_tmpa_int` is the number of dots of the dotted line.

```

2378 \bool_if:NTF \l_@@_initial_open_bool
2379 {
2380   \bool_if:NTF \l_@@_final_open_bool
2381   {
2382     \int_set:Nn \l_tmpa_int
2383     { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
2384   }
2385   {
2386     \int_set:Nn \l_tmpa_int
2387     {
2388       \dim_ratio:nn
2389       { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
2390       \l_@@_inter_dots_dim
2391     }
2392   }
2393 }
2394 {
2395   \bool_if:NTF \l_@@_final_open_bool
2396   {
2397     \int_set:Nn \l_tmpa_int
2398     {
2399       \dim_ratio:nn
2400       { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
2401       \l_@@_inter_dots_dim
2402     }
2403   }
2404   {
2405     \int_set:Nn \l_tmpa_int
2406     {
2407       \dim_ratio:nn
2408       { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
2409       \l_@@_inter_dots_dim
2410     }
2411   }
2412 }

```

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

```

2413 \dim_set:Nn \l_tmpa_dim
2414 {
2415   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2416   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
2417 }
2418 \dim_set:Nn \l_tmpb_dim
2419 {
2420   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
2421   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
2422 }

```

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`.

```

2423 \int_set:Nn \l_tmpb_int
2424 {
2425   \bool_if:NTF \l_@@_initial_open_bool
2426   { \bool_if:NTF \l_@@_final_open_bool 1 0 }
2427   { \bool_if:NTF \l_@@_final_open_bool 2 1 }
2428 }

```

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.

```

2429 \dim_gadd:Nn \l_@@_x_initial_dim
2430 {
2431   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2432   \dim_ratio:nn
2433   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
2434   { 2 \l_@@_l_dim }
2435   * \l_tmpb_int
2436 }
2437 \dim_gadd:Nn \l_@@_y_initial_dim
2438 {
2439   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
2440   \dim_ratio:nn
2441   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
2442   { 2 \l_@@_l_dim }
2443   * \l_tmpb_int
2444 }
2445 \pgf@relevantforpicturesizefalse
2446 \int_step_inline:nnn 0 \l_tmpa_int
2447 {
2448   \pgfpathcircle
2449   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
2450   { \l_@@_radius_dim }
2451   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
2452   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
2453 }
2454 \pgfusepathqfill
2455 }

```

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 for 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.

```

2456 \AtBeginDocument
2457 {
2458   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
2459   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
2460   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
2461   {
2462     \int_compare:nNnTF \c@jCol = 0
2463     { \@@_error:nn { in~first~col } \Ldots }
2464     {
2465       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
2466       { \@@_error:nn { in~last~col } \Ldots }
2467       {
2468         \@@_instruction_of_type:nn { Ldots }
2469         { #1 , down = #2 , up = #3 }
2470       }
2471     }
2472   }

```

```

2471     }
2472     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ldots }
2473     \bool_gset_true:N \g_@@_empty_cell_bool
2474 }

2475 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
2476 {
2477     \int_compare:nNnTF \c@jCol = 0
2478     { \@@_error:nn { in~first~col } \Cdots }
2479     {
2480         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
2481         { \@@_error:nn { in~last~col } \Cdots }
2482         {
2483             \@@_instruction_of_type:nn { Cdots }
2484             { #1 , down = #2 , up = #3 }
2485         }
2486     }
2487     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_cdots }
2488     \bool_gset_true:N \g_@@_empty_cell_bool
2489 }

2490 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
2491 {
2492     \int_compare:nNnTF \c@iRow = 0
2493     { \@@_error:nn { in~first~row } \Vdots }
2494     {
2495         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
2496         { \@@_error:nn { in~last~row } \Vdots }
2497         {
2498             \@@_instruction_of_type:nn { Vdots }
2499             { #1 , down = #2 , up = #3 }
2500         }
2501     }
2502     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_vdots }
2503     \bool_gset_true:N \g_@@_empty_cell_bool
2504 }

2505 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
2506 {
2507     \int_case:nnF \c@iRow
2508     {
2509         0 { \@@_error:nn { in~first~row } \Ddots }
2510         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
2511     }
2512     {
2513         \int_case:nnF \c@jCol
2514         {
2515             0 { \@@_error:nn { in~first~col } \Ddots }
2516             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
2517         }
2518         {
2519             \@@_instruction_of_type:nn { Ddots }
2520             { #1 , down = #2 , up = #3 }
2521         }
2522     }
2523 }
2524 \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ddots }
2525 \bool_gset_true:N \g_@@_empty_cell_bool
2526 }

```

```

2527 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
2528 {
2529   \int_case:nnF \c@iRow
2530   {
2531     0 { \@@_error:nn { in~first~row } \Iddots }
2532     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
2533   }
2534   {
2535     \int_case:nnF \c@jCol
2536     {
2537       0 { \@@_error:nn { in~first~col } \Iddots }
2538       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
2539     }
2540     {
2541       \@@_instruction_of_type:nn { Iddots }
2542       { #1 , down = #2 , up = #3 }
2543     }
2544   }
2545   }
2546   \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_iddots }
2547   \bool_gset_true:N \g_@@_empty_cell_bool
2548 }
2549 }

```

End of the \AtBeginDocument.

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

```

2550 \cs_new_protected:Npn \@@_Hspace:
2551 {
2552   \bool_gset_true:N \g_@@_empty_cell_bool
2553   \hspace
2554 }

```

In the environment {NiceArray}, the command \multicolumn will be linked to the following command \@@_multicolumn:nnn.

```

2555 \cs_set_eq:NN \@@_old_multicolumn \multicolumn
2556 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2557 {
2558   \@@_old_multicolumn { #1 } { #2 } { #3 }

```

The \peek_remove_spaces:n is mandatory.

```

2559 \peek_remove_spaces:n
2560 {
2561   \int_compare:nNnT #1 > 1
2562   {
2563     \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2564     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2565     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2566     \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2567     {
2568       { \int_use:N \c@iRow }
2569       { \int_use:N \c@jCol }
2570       { \int_use:N \c@iRow }
2571       { \int_eval:n { \c@jCol + #1 - 1 } }
2572     }
2573   }
2574   \int_gadd:Nn \c@jCol { #1 - 1 }
2575 }
2576 }

```

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`.

```

2577 \cs_new:Npn \@@_Hdotsfor:
2578 {
2579   \multicolumn { 1 } { C } { }
2580   \@@_Hdotsfor_i
2581 }

```

The command `\@@_Hdotsfor_i` is defined with the tools of `xparse` 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:`).

```

2582 \AtBeginDocument
2583 {
2584   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
2585   \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.

```

2586   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
2587   {
2588     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
2589     {
2590       \@@_Hdotsfor:nnnn
2591       { \int_use:N \c@iRow }
2592       { \int_use:N \c@jCol }
2593       { #2 }
2594       {
2595         #1 , #3 ,
2596         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
2597       }
2598     }
2599     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } }
2600   }
2601 }

```

Enf of `\AtBeginDocument`.

```

2602 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
2603 {
2604   \bool_set_false:N \l_@@_initial_open_bool
2605   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

2606   \int_set:Nn \l_@@_initial_i_int { #1 }
2607   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

2608   \int_compare:nNnTF #2 = 1
2609   {
2610     \int_set:Nn \l_@@_initial_j_int 1
2611     \bool_set_true:N \l_@@_initial_open_bool
2612   }
2613   {
2614     \cs_if_exist:cTF
2615     {
2616       pgf @ sh @ ns @ \@@_env:
2617       - \int_use:N \l_@@_initial_i_int
2618       - \int_eval:n { #2 - 1 }
2619     }
2620     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
2621     {
2622       \int_set:Nn \l_@@_initial_j_int { #2 }
2623       \bool_set_true:N \l_@@_initial_open_bool
2624     }
2625   }
2626   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol

```

```

2627 {
2628   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
2629   \bool_set_true:N \l_@@_final_open_bool
2630 }
2631 {
2632   \cs_if_exist:cTF
2633   {
2634     pgf @ sh @ ns @ \@@_env:
2635     - \int_use:N \l_@@_final_i_int
2636     - \int_eval:n { #2 + #3 }
2637   }
2638   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
2639   {
2640     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
2641     \bool_set_true:N \l_@@_final_open_bool
2642   }
2643 }
2644 \group_begin:
2645 \int_compare:nNnTF { #1 } = 0
2646 { \color { nicematrix-first-row } }
2647 {
2648   \int_compare:nNnT { #1 } = \g_@@_row_total_int
2649   { \color { nicematrix-last-row } }
2650 }
2651 \keys_set:nn { NiceMatrix / xdots } { #4 }
2652 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2653 \@@_actually_draw_Ldots:
2654 \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).

```

2655   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
2656   { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
2657 }

2658 \AtBeginDocument
2659 {
2660   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
2661   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
2662   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
2663   {
2664     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
2665     {
2666       \@@_Vdotsfor:nnnn
2667       { \int_use:N \c@iRow }
2668       { \int_use:N \c@jCol }
2669       { #2 }
2670       {
2671         #1 , #3 ,
2672         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
2673       }
2674     }
2675   }
2676 }

```

Enf of `\AtBeginDocument`.

```

2677 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
2678 {
2679   \bool_set_false:N \l_@@_initial_open_bool
2680   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```
2681 \int_set:Nn \l_@@_initial_j_int { #2 }
2682 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

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

```
2683 \int_compare:nNnTF #1 = 1
2684 {
2685   \int_set:Nn \l_@@_initial_i_int 1
2686   \bool_set_true:N \l_@@_initial_open_bool
2687 }
2688 {
2689   \cs_if_exist:cTF
2690   {
2691     pgf @ sh @ ns @ \@@_env:
2692     - \int_eval:n { #1 - 1 }
2693     - \int_use:N \l_@@_initial_j_int
2694   }
2695   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
2696   {
2697     \int_set:Nn \l_@@_initial_i_int { #1 }
2698     \bool_set_true:N \l_@@_initial_open_bool
2699   }
2700 }
2701 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
2702 {
2703   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
2704   \bool_set_true:N \l_@@_final_open_bool
2705 }
2706 {
2707   \cs_if_exist:cTF
2708   {
2709     pgf @ sh @ ns @ \@@_env:
2710     - \int_eval:n { #1 + #3 }
2711     - \int_use:N \l_@@_final_j_int
2712   }
2713   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
2714   {
2715     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
2716     \bool_set_true:N \l_@@_final_open_bool
2717   }
2718 }
2719 \group_begin:
2720 \int_compare:nNnTF { #2 } = 0
2721 { \color { nicematrix-first-col } }
2722 {
2723   \int_compare:nNnT { #2 } = \g_@@_col_total_int
2724   { \color { nicematrix-last-col } }
2725 }
2726 \keys_set:nn { NiceMatrix / xdots } { #4 }
2727 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2728 \@@_actually_draw_Vdots:
2729 \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).

```
2730 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
2731 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
2732 }
```

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

The command will exit three levels of groups (only two in `{NiceTabular}` because there is not the group of the math mode to exit) in order to execute the command

`“\box_rotate:Nn \l_@@_cell_box { 90 }”`

just after the construction of the box `\l_@@_cell_box`.

```

2733 \cs_new_protected:Npn \@@_rotate:
2734 {
2735   \bool_if:NTF \l_@@_NiceTabular_bool
2736     { \group_insert_after:N \@@_rotate_ii: }
2737     { \group_insert_after:N \@@_rotate_i: }
2738 }
2739 \cs_new_protected:Npn \@@_rotate_i: { \group_insert_after:N \@@_rotate_ii: }
2740 \cs_new_protected:Npn \@@_rotate_ii: { \group_insert_after:N \@@_rotate_iii: }
2741 \cs_new_protected:Npn \@@_rotate_iii:
2742 {
2743   \box_rotate:Nn \l_@@_cell_box { 90 }

```

If we are in the last row, we want all the boxes composed with the command `\rotate` aligned upwards.

```

2744   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
2745   {
2746     \vbox_set_top:Nn \l_@@_cell_box
2747     {
2748       \vbox_to_zero:n { }

```

0.8 ex will be the distance between the principal part of the array and our element (which is composed with `\rotate`).

```

2749       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
2750       \box_use:N \l_@@_cell_box
2751     }
2752   }
2753 }

```

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).³⁹

```

2754 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
2755 { \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).

```

2756 \AtBeginDocument
2757 {
2758   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
2759   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
2760   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
2761   {
2762     \group_begin:
2763     \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
2764     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2765     \use:x
2766     {
2767       \@@_line_i:nn
2768       { \@@_double_int_eval:n #2 \q_stop }
2769       { \@@_double_int_eval:n #3 \q_stop }

```

³⁹Indeed, we want that the user may use the command `\line` in `code-after` with LaTeX counters in the arguments — with the command `\value`.

```

2770     }
2771     \group_end:
2772 }
2773 }
2774 \cs_new_protected:Npn \@@_line_i:nn #1 #2
2775 {
2776     \bool_set_false:N \l_@@_initial_open_bool
2777     \bool_set_false:N \l_@@_final_open_bool
2778     \bool_if:nTF
2779     {
2780         \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
2781         ||
2782         \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
2783     }
2784     {
2785         \@@_error:nnn { unknown~cell~for~line~in~code~after } { #1 } { #2 }
2786     }
2787     { \@@_draw_line_ii:nn { #1 } { #2 } }
2788 }
2789 \AtBeginDocument
2790 {
2791     \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
2792     {

```

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:`.

```

2793     \c_@@_pgfortikzpicture_tl
2794     \@@_draw_line_iii:nn { #1 } { #2 }
2795     \c_@@_endpgfortikzpicture_tl
2796 }
2797 }

```

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

```

2798 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
2799 {
2800     \pgfrememberpicturepositiononpagetrue
2801     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
2802     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2803     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2804     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
2805     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2806     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2807     \@@_draw_line:
2808 }

```

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).

Commands available in the code-before

In the beginning of the code-before, the command `\@@_rowcolor:nn` will be linked to `\rowcolor` and the command `\@@_columncolor:nn` to `\columncolor`.

```

2809 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
2810 {
2811     \tl_set:Nn \l_tmpa_tl { #1 }
2812     \tl_set:Nn \l_tmpb_tl { #2 }
2813 }

```

Here an example : \@@_rowcolor {red!15} {1,3,5-7,10-}

```

2814 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
2815 {
2816   \tl_if_blank:nF { #2 }
2817   {
2818     \pgfpicture
2819     \pgf@relevantforpicturesizefalse
2820     \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }

```

\l_tmpa_dim is the x -value of the right side of the rows.

```

2821   \@@_qpoint:n { col - 1 }
2822   \int_compare:nNnTF \l_@@_first_col_int = 0
2823   { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
2824   { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
2825   \@@_qpoint:n { col - \@@_succ:n \c@jCol }
2826   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
2827   \clist_map_inline:nn { #3 }
2828   {
2829     \tl_set:Nn \l_tmpa_tl { ##1 }
2830     \tl_if_in:NnTF \l_tmpa_tl { - }
2831     { \@@_cut_on_hyphen:w ##1 \q_stop }
2832     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
2833     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
2834     \tl_if_empty:NT \l_tmpb_tl
2835     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
2836     \int_compare:nNnT \l_tmpb_tl > \c@iRow
2837     { \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.

```

2838     \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
2839     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
2840     \@@_qpoint:n { row - \l_tmpa_tl }
2841     \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
2842     \pgfpathrectanglecorners
2843     { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
2844     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
2845   }
2846   \pgfusepathqfill
2847   \endpgfpicture
2848 }
2849 }

```

Here an example : \@@_columncolor:nn {red!15} {1,3,5-7,10-}

```

2850 \NewDocumentCommand \@@_columncolor { 0 { } m m }
2851 {
2852   \tl_if_blank:nF { #2 }
2853   {
2854     \pgfpicture
2855     \pgf@relevantforpicturesizefalse
2856     \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
2857     \@@_qpoint:n { row - 1 }

```

\l_tmpa_dim is the y -value of the top of the columns et \l_tmpb_dim is the y -value of the bottom.

```

2858     \dim_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
2859     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
2860     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
2861     \clist_map_inline:nn { #3 }
2862     {
2863       \tl_set:Nn \l_tmpa_tl { ##1 }
2864       \tl_if_in:NnTF \l_tmpa_tl { - }
2865       { \@@_cut_on_hyphen:w ##1 \q_stop }
2866       { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
2867       \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
2868       \tl_if_empty:NT \l_tmpb_tl

```

```

2869         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
2870 \int_compare:nNnT \l_tmpb_tl > \c@jCol
2871         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

Now, the numbers of both columns are in \l_tmpa_tl and \l_tmpb_tl.

```

2872         \@@_qpoint:n { col - \l_tmpa_tl }
2873 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
2874     { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
2875     { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
2876 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
2877 \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
2878 \pgfpathrectanglecorners
2879     { \pgfpoint \l_tmpc_dim \l_tmpa_dim }
2880     { \pgfpoint \l_tmpd_dim \l_tmpb_dim }
2881 }
2882 \pgfusepathqfill
2883 \endpgfpicture
2884 }
2885 }

```

Here an example : \@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}

```

2886 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
2887 {
2888     \tl_if_blank:nF { #2 }
2889     {
2890         \pgfpicture
2891         \pgf@relevantforpicturesizefalse
2892         \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
2893         \clist_map_inline:nn { #3 }
2894         {
2895             \@@_cut_on_hyphen:w ##1 \q_stop
2896             \@@_qpoint:n { row - \l_tmpa_tl }
2897             \bool_lazy_and:nnT
2898                 { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
2899                 { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
2900             {
2901                 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
2902                 \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
2903                 \dim_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
2904                 \@@_qpoint:n { col - \l_tmpb_tl }
2905                 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpb_tl
2906                     { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
2907                     { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
2908                 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
2909                 \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
2910                 \pgfpathrectanglecorners
2911                     { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
2912                     { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
2913             }
2914         }
2915         \pgfusepathqfill
2916         \endpgfpicture
2917     }
2918 }

```

Here an example : \@@_rectanglecolor{red!15}{2-3}{5-6}

```

2919 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
2920 {
2921     \tl_if_blank:nF { #2 }
2922     {
2923         \pgfpicture
2924         \pgf@relevantforpicturesizefalse
2925         \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }

```

```

2926 \@@_cut_on_hyphen:w #3 \q_stop
2927 \bool_lazy_and:nnT
2928 { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
2929 { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
2930 {
2931   \@@_qpoint:n { row - \l_tmpa_tl }
2932   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
2933   \@@_qpoint:n { col - \l_tmpb_tl }
2934   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpb_tl
2935     { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
2936     { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
2937   \@@_cut_on_hyphen:w #4 \q_stop
2938   \int_compare:nNnT \l_tmpa_tl > \c@iRow
2939     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
2940   \int_compare:nNnT \l_tmpb_tl > \c@jCol
2941     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
2942   \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
2943   \dim_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
2944   \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
2945   \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
2946   \pgfpathrectanglecorners
2947     { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
2948     { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
2949   \pgfusepathqfill
2950 }
2951 \endpgfpicture
2952 }
2953 }

```

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`.

```

2954 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
2955 {
2956   \int_step_inline:nnn { #2 } { \int_use:N \c@iRow }
2957   {
2958     \int_if_odd:nTF { ##1 }
2959       { \@@_rowcolor [ #1 ] { #3 } }
2960       { \@@_rowcolor [ #1 ] { #4 } }
2961     { ##1 }
2962   }
2963 }

2964 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
2965 {
2966   \int_step_inline:nn { \int_use:N \c@iRow }
2967   {
2968     \int_step_inline:nn { \int_use:N \c@jCol }
2969     {
2970       \int_if_even:nTF { #####1 + ##1 }
2971         { \@@_cellcolor [ #1 ] { #2 } }
2972         { \@@_cellcolor [ #1 ] { #3 } }
2973       { ##1 - #####1 }
2974     }
2975   }
2976 }

```

The vertical rules

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential

exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
2977 \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`.

```
2978 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
2979 {
2980   \int_compare:nNnTF \l_@@_first_col_int = 0
2981     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
2982     {
2983       \int_compare:nNnTF \c@jCol = 0
2984         {
2985           \int_compare:nNnF \c@iRow = { -1 }
2986             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
2987         }
2988       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
2989     }
2990 }
```

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.

```
2991 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
2992 {
2993   \int_compare:nNnF \c@iRow = 0
2994     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
2995 }
```

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`).

In fact, independently of `\OnlyMainNiceMatrix`, which is a convenience given to the user, we have to modify the behaviour of the standard specifier “|”.

Remark first that the natural way to do that would be to redefine the specifier “|” with `\newcolumnmtype`:

```
\newcolumnmtype { | } { ! { \OnlyMainNiceMatrix \vline } }
```

However, this code fails if the user uses `\DefineShortVerb{\\}` of `fancyvrb`. Moreover, it would not be able to deal correctly with two consecutive specifiers “|” (in a preamble like `ccc|ccc`).

That's why we have done a redefinition of the macro `\@arrayrule` of `array` and this redefinition will add `\@@_vline`: instead of `\vline` in the preamble (that definition is in the beginning of `{NiceArrayWithDelims}`).

Here is the definition of `\@@_vline`:. This definition *must* be protected because you don't want that macro expanded during the construction of the preamble (the tests in `\@@_OnlyMainNiceMatrix:n` must be effective in each row and not once for all when the preamble is constructed). 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.

```
2996 \cs_new_protected:Npn \@@_vline:
2997 { \@@_OnlyMainNiceMatrix:n { { \CT@arc@ \vline } } }
```

The command `\@@_draw_vlines` will be executed when the user uses the option `vlines` (which draws all the vlines of the array).

```
2998 \cs_new_protected:Npn \@@_draw_vlines:
2999 {
3000   \group_begin:
```

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 when `colortbl` is not loaded.

```

3001 \CT@arc@
3002 \pgfpicture
3003 \pgfrememberpicturepositiononpagetrue
3004 \pgf@relevantforpicturesizefalse
3005 \pgfsetlinewidth \arrayrulewidth
3006 \pgfsetrectcap
3007 \@@_qpoint:n { row - 1 }
3008 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3009 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
3010 \dim_set_eq:NN \l_tmpb_dim \pgf@y

```

Now, we can draw the vertical rules with a loop.

```

3011 \int_step_inline:nnn
3012 { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
3013 { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
3014 {
3015   \@@_qpoint:n { col - ##1 }
3016   \dim_set_eq:NN \l_tmpc_dim \pgf@x
3017   \pgfpathmoveto { \pgfpoint \l_tmpc_dim \l_tmpa_dim }
3018   \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
3019 }
3020 \pgfusepathqstroke
3021 \endpgfpicture
3022 \group_end:
3023 }

```

The key hvlines

The key hvlines

```

3024 \cs_new_protected:Npn \@@_draw_hlines:
3025 {
3026   \pgfpicture
3027   \CT@arc@
3028   \pgfrememberpicturepositiononpagetrue
3029   \pgf@relevantforpicturesizefalse
3030   \pgfsetlinewidth \arrayrulewidth
3031   \int_step_inline:nnn
3032   { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
3033   { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
3034   {
3035     \@@_qpoint:n { row - ##1 }
3036     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3037     \pgfpathmoveto { \pgfpoint \pgf@x \pgf@y }
3038     \@@_qpoint:n { col - \@@_succ:n { \c@jCol } }
3039     \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \arrayrulewidth }
3040     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
3041   }
3042   \pgfusepathqstroke
3043   \endpgfpicture
3044 }

```

Since version 4.1, the key `hvlines` is no longer a mere alias for the conjunction of `hlines` and `vlines`. Indeed, with `hvlines`, the vertical and horizontal rules are *not* drawn within the blocks (created by `\Block`) nor within the “virtual blocks” (corresponding to the dotted lines drawn by `\Cdots`, `\Vdots`, etc.).

```

3045 \cs_new_protected:Npn \@@_draw_hvlines:
3046 {
3047   \bool_lazy_and:nnTF
3048   { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }

```

```

3049 { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
3050 \@@_draw_hvlines_i:
3051 \@@_draw_hvlines_ii:
3052 }

```

This version is only for efficiency. The general case (in \@@_draw_hvlines_ii:) does the job in all case (but slower).

```

3053 \cs_new_protected:Npn \@@_draw_hvlines_i:
3054 {
3055   \@@_draw_hlines:
3056   \@@_draw_vlines:
3057 }

```

Now, the general case, where there are blocks or dots in the array.

```

3058 \cs_new_protected:Npn \@@_draw_hvlines_ii:
3059 {
3060   \group_begin:
3061   \CT@arc@

```

The horizontal rules.

```

3062   \int_step_variable:nnNn
3063   { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
3064   { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
3065   \l_tmpa_tl
3066   {
3067     \int_step_variable:nnNn \c@jCol \l_tmpb_tl
3068     {

```

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.

```

3069       \bool_gset_true:N \g_tmpa_bool
3070       \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3071       { \@@_test_if_hline_in_block:nnnn ##1 }
3072       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
3073       { \@@_test_if_hline_in_block:nnnn ##1 }
3074       \bool_if:NT \l_@@_hvlines_except_corners_bool
3075       {
3076         \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
3077         {
3078           \seq_if_in:NxT
3079           \l_@@_empty_corner_cells_seq
3080           { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
3081           { \bool_set_false:N \g_tmpa_bool }
3082         }
3083         {
3084           \seq_if_in:NxT
3085           \l_@@_empty_corner_cells_seq
3086           { \l_tmpa_tl - \l_tmpb_tl }
3087           {
3088             \int_compare:nNnTF \l_tmpa_tl = 1
3089             { \bool_set_false:N \g_tmpa_bool }
3090             {
3091               \seq_if_in:NxT
3092               \l_@@_empty_corner_cells_seq
3093               { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
3094               { \bool_set_false:N \g_tmpa_bool }
3095             }
3096           }
3097         }
3098       }
3099       \bool_if:NT \g_tmpa_bool

```

```

3100         {
3101             \pgfpicture
3102             \pgfrememberpicturepositiononpagetrue
3103             \pgf@relevantforpicturesizefalse
3104             \pgfsetlinewidth \arrayrulewidth
3105             \pgfsetrectcap
3106             \@@_qpoint:n { row - \l_tmpa_tl }
3107             \dim_set_eq:NN \l_tmpb_dim \pgf@y
3108             \@@_qpoint:n { col - \l_tmpb_tl }
3109             \dim_set_eq:NN \l_tmpa_dim \pgf@x
3110             \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
3111             \dim_set_eq:NN \l_tmpc_dim \pgf@x
3112             \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3113             \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
3114             \pgfusepathqstroke
3115             \endpgfpicture
3116         }
3117     }
3118 }

```

Now, the vertical rules.

```

3119 \int_step_variable:nNn \c@iRow \l_tmpa_tl
3120 {
3121     \int_step_variable:nnNn
3122     { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
3123     { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
3124     \l_tmpb_tl
3125     {

```

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.

```

3126     \bool_gset_true:N \g_tmpa_bool
3127     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3128     { \@@_test_if_vline_in_block:nnnn ##1 }
3129     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
3130     { \@@_test_if_vline_in_block:nnnn ##1 }
3131     \bool_if:NT \l_@@_hvlines_except_corners_bool
3132     {
3133         \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
3134         {
3135             \seq_if_in:NxT
3136             \l_@@_empty_corner_cells_seq
3137             { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
3138             { \bool_set_false:N \g_tmpa_bool }
3139         }
3140         {
3141             \seq_if_in:NxT
3142             \l_@@_empty_corner_cells_seq
3143             { \l_tmpa_tl - \l_tmpb_tl }
3144             {
3145                 \int_compare:nNnTF \l_tmpb_tl = 1
3146                 { \bool_set_false:N \g_tmpa_bool }
3147                 {
3148                     \seq_if_in:NxT
3149                     \l_@@_empty_corner_cells_seq
3150                     { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
3151                     { \bool_set_false:N \g_tmpa_bool }
3152                 }
3153             }
3154         }
3155     }
3156     \bool_if:NT \g_tmpa_bool

```

```

3157         {
3158             \pgfpicture
3159             \pgfrememberpicturepositiononpagetrue
3160             \pgf@relevantforpicturesizefalse
3161             \pgfsetlinewidth \arrayrulewidth
3162             \pgfsetrectcap
3163             \@@_qpoint:n { row - \l_tmpa_tl }
3164             \dim_set_eq:NN \l_tmpb_dim \pgf@y
3165             \@@_qpoint:n { col - \l_tmpb_tl }
3166             \dim_set_eq:NN \l_tmpa_dim \pgf@x
3167             \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
3168             \dim_set_eq:NN \l_tmpc_dim \pgf@y
3169             \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3170             \pgfpathlineto { \pgfpoint \l_tmpa_dim \l_tmpc_dim }
3171             \pgfusepathqstroke
3172             \endpgfpicture
3173         }
3174     }
3175 }

```

The group was for the color of the rules.

```

3176     \group_end:
3177     \seq_gclear:N \g_@@_pos_of_xdots_seq
3178 }

```

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.

```

3179 \cs_new_protected:Npn \@@_test_if_hline_in_block:nnnn #1 #2 #3 #4
3180 {
3181     \int_compare:nNnT \l_tmpa_tl > { #1 }
3182     {
3183         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
3184         {
3185             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
3186             {
3187                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
3188                 { \bool_gset_false:N \g_tmpa_bool }
3189             }
3190         }
3191     }
3192 }

```

The same for vertical rules.

```

3193 \cs_new_protected:Npn \@@_test_if_vline_in_block:nnnn #1 #2 #3 #4
3194 {
3195     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
3196     {
3197         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
3198         {
3199             \int_compare:nNnT \l_tmpb_tl > { #2 }
3200             {
3201                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
3202                 { \bool_gset_false:N \g_tmpa_bool }
3203             }
3204         }
3205     }
3206 }

```

The key `hvlines-except-corners`

```

3207 \cs_new_protected:Npn \@@_draw_hvlines_except_corners:
3208 {

```

The sequence `\l_@@_empty_corner_cells_seq` will be the sequence of all the cells empty (and not in a block) considered in the corners of the array.

```

3209 \seq_clear_new:N \l_@@_empty_corner_cells_seq
3210 \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol
3211 \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1
3212 \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol
3213 \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1
3214 \@@_draw_hvlines_ii:
3215 }

```

“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 step 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.

```

3216 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
3217 {

```

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.

```

3218 \bool_set_false:N \l_tmpa_bool
3219 \int_zero_new:N \l_@@_last_empty_row_int
3220 \int_step_inline:nnnn { #1 } { #3 } { #5 }
3221 {
3222 \@@_test_if_cell_in_a_block:nx { ##1 } { \int_eval:n { #2 } }
3223 \bool_if:nTF
3224 {
3225 \cs_if_exist_p:c
3226 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
3227 ||
3228 \l_tmpb_bool
3229 }
3230 { \bool_set_true:N \l_tmpa_bool }
3231 {
3232 \bool_if:NF \l_tmpa_bool
3233 { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
3234 }
3235 }

```

Now, you determine the last empty cell in the row of number 1.

```

3236 \bool_set_false:N \l_tmpa_bool
3237 \int_zero_new:N \l_@@_last_empty_column_int
3238 \int_step_inline:nnnn { #2 } { #4 } { #6 }
3239 {
3240 \@@_test_if_cell_in_a_block:xn { \int_eval:n { #1 } } { ##1 }
3241 \bool_if:nTF
3242 {
3243 \cs_if_exist_p:c
3244 { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
3245 || \l_tmpb_bool
3246 }
3247 { \bool_set_true:N \l_tmpa_bool }
3248 {
3249 \bool_if:NF \l_tmpa_bool

```

```

3250         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
3251     }
3252 }

```

Now, we loop over the rows.

```

3253 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
3254 {

```

We treat the row number ##1 with another loop.

```

3255     \bool_set_false:N \l_tmpa_bool
3256     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
3257     {
3258         \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
3259         \bool_if:nTF
3260         {
3261             \cs_if_exist_p:c
3262             { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
3263             || \l_tmpb_bool
3264         }
3265         { \bool_set_true:N \l_tmpa_bool }
3266         {
3267             \bool_if:NF \l_tmpa_bool
3268             {
3269                 \int_set:Nn \l_@@_last_empty_column_int { ####1 }
3270                 \seq_put_right:Nn
3271                 \l_@@_empty_corner_cells_seq
3272                 { ##1 - ####1 }
3273             }
3274         }
3275     }
3276 }
3277 }

```

The following macro tests whether if a cell is in (at least) one of the blocks of the array. The flag `\l_tmpb_bool` will be raised if the cell #1-#2 is in a block.

```

3278 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
3279 {
3280     \bool_set_false:N \l_tmpb_bool
3281     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3282     { \@@_test_if_cell_in_block:nnnnnn { #1 } { #2 } ##1 }
3283 }
3284 \cs_generate_variant:Nn \@@_test_if_cell_in_a_block:nn { nx , xn }
3285 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6
3286 {
3287     \bool_if:nT
3288     {
3289         \int_compare_p:n { #3 <= #1 }
3290         && \int_compare_p:n { #1 <= #5 }
3291         && \int_compare_p:n { #4 <= #2 }
3292         && \int_compare_p:n { #2 <= #6 }
3293     }
3294     { \bool_set_true:N \l_tmpb_bool }
3295 }

```

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`.

```

3296 \cs_new:Npn \@@_hdottedline:
3297 {
3298   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
3299   \@@_hdottedline_i:
3300 }

```

On the other side, the following command should be protected.

```

3301 \cs_new_protected:Npn \@@_hdottedline_i:
3302 {

```

We write in the code-after the instruction that will eventually 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).

```

3303   \tl_gput_right:Nx \g_@@_internal_code_after_tl
3304   { \@@_hdottedline:n { \int_use:N \c@iRow } }
3305 }

```

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.

```

3306 \AtBeginDocument
3307 {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”.

```

3308   \cs_new_protected:Npx \@@_hdottedline:n #1
3309   {
3310     \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
3311     \bool_set_true:N \exp_not:N \l_@@_final_open_bool
3312     \c_@@_pgfortikzpicture_tl
3313     \@@_hdottedline_i:n { #1 }
3314     \c_@@_endpgfortikzpicture_tl
3315   }
3316 }

```

The following command *must* be protected since it is used in the construction of `\@@_hdottedline:n`.

```

3317 \cs_new_protected:Npn \@@_hdottedline_i:n #1
3318 {
3319   \pgfrememberpicturerepositiononpagetrue
3320   \@@_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).

```

3321   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3322   \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3323   \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}`

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

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}`

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

```

3324 \@@_qpoint:n { col - 1 }
3325 \dim_set:Nn \l_@@_x_initial_dim
3326 {
3327   \pgf@x +
3328   \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep
3329   - \l_@@_left_margin_dim
3330 }
3331 \@@_qpoint:n { col - \@@_succ:n \c@jCol }
3332 \dim_set:Nn \l_@@_x_final_dim
3333 {
3334   \pgf@x -
3335   \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep
3336   + \l_@@_right_margin_dim
3337 }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by $0.5 \text{ } \l_@@_inter_dots_dim$ is *ad hoc* for a better result.

```

3338 \tl_set:Nn \l_tmpa_tl { ( }
3339 \tl_if_eq:NNTF \l_@@_left_delim_tl \l_tmpa_tl
3340 { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
3341 \tl_set:Nn \l_tmpa_tl { ) }
3342 \tl_if_eq:NNTF \l_@@_right_delim_tl \l_tmpa_tl
3343 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }

```

As for 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`.

```

3344 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
3345 \@@_draw_line:
3346 }

```

Vertical dotted lines

```

3347 \cs_new_protected:Npn \@@_vdottedline:n #1
3348 {
3349   \bool_set_true:N \l_@@_initial_open_bool
3350   \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”.

```

3351 \bool_if:NTF \c_@@_tikz_loaded_bool
3352 {
3353   \tikzpicture
3354   \@@_vdottedline_i:n { #1 }
3355   \endtikzpicture
3356 }
3357 {
3358   \pgfpicture
3359   \@@_vdottedline_i:n { #1 }
3360   \endpgfpicture
3361 }
3362 }

```

```

3363 \cs_new_protected:Npn \@@_vdottedline_i:n #1
3364 {

```

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.

```

3365 \CT@arc@
3366 \pgfrememberpicturepositiononpagetrue
3367 \@@_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).

```
3368 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
3369 \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
3370 \@@_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.

```
3371 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
3372 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
3373 \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }
```

As for 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`.

```
3374 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
3375 \@@_draw_line:
3376 }
```

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.

```
3377 \bool_new:N \l_@@_block_auto_columns_width_bool
```

As of now, there is only one option available for the environment {NiceMatrixBlock}.

```
3378 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
3379 {
3380   auto-columns-width .code:n =
3381   {
3382     \bool_set_true:N \l_@@_block_auto_columns_width_bool
3383     \dim_gzero_new:N \g_@@_max_cell_width_dim
3384     \bool_set_true:N \l_@@_auto_columns_width_bool
3385   }
3386 }

3387 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
3388 {
3389   \int_gincr:N \g_@@_NiceMatrixBlock_int
3390   \dim_zero:N \l_@@_columns_width_dim
3391   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
3392   \bool_if:NT \l_@@_block_auto_columns_width_bool
3393   {
3394     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
3395     {
3396       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
3397       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
3398     }
3399   }
3400 }
```

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`).

```
3401 {
3402   \bool_if:NT \l_@@_block_auto_columns_width_bool
3403   {
3404     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
3405     \iow_shipout:Nx \@mainaux
```

```

3406     {
3407         \cs_gset:cpn
3408         { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
For technical reasons, we have to include the width of an eventual rule on the right side of the cells.
3409         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
3410     }
3411     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
3412 }
3413 }

```

The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

3414 \cs_generate_variant:Nn \dim_min:nn { v n }
3415 \cs_generate_variant:Nn \dim_max:nn { v n }

```

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`.

```

3416 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
3417 {
3418     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
3419     {
3420         \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
3421         \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
3422         \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
3423         \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
3424     }
3425     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
3426     {
3427         \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
3428         \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
3429         \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
3430         \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
3431     }

```

We begin the two nested loops over the rows and the columns of the array.

```

3432     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
3433     {
3434         \int_step_variable:nnNn
3435         \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.

```

3436     {
3437         \cs_if_exist:cT
3438         { 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.

```

3439     {
3440         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
3441         \dim_set:cn { l_@@_row _ \@@_i: _ min_dim }
3442         { \dim_min:vn { l_@@_row _ \@@_i: _ min_dim } \pgf@y }
3443         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
3444         {
3445             \dim_set:cn { l_@@_column _ \@@_j: _ min_dim }
3446             { \dim_min:vn { l_@@_column _ \@@_j: _ min_dim } \pgf@x }
3447         }

```

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.

```

3448         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
3449         \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
3450         { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
3451         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
3452         {
3453             \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
3454             { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
3455         }
3456     }
3457 }
3458 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

3459 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
3460 {
3461     \dim_compare:nNnT
3462     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
3463     {
3464         \@@_qpoint:n { row - \@@_i: - base }
3465         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
3466         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
3467     }
3468 }
3469 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
3470 {
3471     \dim_compare:nNnT
3472     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
3473     {
3474         \@@_qpoint:n { col - \@@_j: }
3475         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@x
3476         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@x
3477     }
3478 }
3479 }

```

Here is the command \@@_create_medium_nodes:. When this command is used, the “medium nodes” are created.

```

3480 \cs_new_protected:Npn \@@_create_medium_nodes:
3481 {
3482     \pgfpicture
3483     \pgfrememberpicturepositiononpagetrue
3484     \pgf@relevantforpicturesizefalse
3485     \@@_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”.

```

3486     \tl_set:Nn \l_@@_suffix_tl { -medium }
3487     \@@_create_nodes:
3488     \endpgfpicture
3489 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁴⁰. 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:`.

```

3490 \cs_new_protected:Npn \@@_create_large_nodes:
3491 {
3492     \pgfpicture
3493     \pgfrememberpicturepositiononpagetrue
3494     \pgf@relevantforpicturesizefalse
3495     \@@_computations_for_medium_nodes:
3496     \@@_computations_for_large_nodes:
3497     \tl_set:Nn \l_@@_suffix_tl { - large }
3498     \@@_create_nodes:
3499     \endpgfpicture
3500 }

3501 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
3502 {
3503     \pgfpicture
3504     \pgfrememberpicturepositiononpagetrue
3505     \pgf@relevantforpicturesizefalse
3506     \@@_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”.

```

3507     \tl_set:Nn \l_@@_suffix_tl { - medium }
3508     \@@_create_nodes:
3509     \@@_computations_for_large_nodes:
3510     \tl_set:Nn \l_@@_suffix_tl { - large }
3511     \@@_create_nodes:
3512     \endpgfpicture
3513 }

```

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.

```

3514 \cs_new_protected:Npn \@@_computations_for_large_nodes:
3515 {
3516     \int_set:Nn \l_@@_first_row_int 1
3517     \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`.

```

3518     \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
3519     {
3520         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
3521         {
3522             (
3523                 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
3524                 \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
3525             )
3526             / 2
3527         }
3528         \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }

```

⁴⁰If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

3529     { l_@@_row\_@@_i: _min_dim }
3530 }
3531 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
3532 {
3533     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
3534     {
3535         (
3536             \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
3537             \dim_use:c
3538                 { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
3539         )
3540         / 2
3541     }
3542     \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
3543     { l_@@_column _ \@@_j: _ max _ dim }
3544 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

3545     \dim_sub:cn
3546     { l_@@_column _ 1 _ min _ dim }
3547     \l_@@_left_margin_dim
3548     \dim_add:cn
3549     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
3550     \l_@@_right_margin_dim
3551 }

```

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).

```

3552 \cs_new_protected:Npn \@@_create_nodes:
3553 {
3554     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
3555     {
3556         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
3557         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

3558         \@@_pgf_rect_node:nnnnn
3559         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
3560         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
3561         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
3562         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
3563         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
3564         \str_if_empty:NF \l_@@_name_str
3565         {
3566             \pgfnodealias
3567             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
3568             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
3569         }
3570     }
3571 }

```

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 .

```

3572     \seq_mapthread_function:NNN
3573     \g_@@_multicolumn_cells_seq
3574     \g_@@_multicolumn_sizes_seq
3575     \@@_node_for_multicolumn:nn
3576 }

```

```

3577 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
3578 {
3579     \cs_set:Npn \@@_i: { #1 }
3580     \cs_set:Npn \@@_j: { #2 }
3581 }

```

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”).

```

3582 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
3583 {
3584     \@@_extract_coords_values: #1 \q_stop
3585     \@@_pgf_rect_node:nnnnn
3586     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
3587     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
3588     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
3589     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
3590     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
3591     \str_if_empty:NF \l_@@_name_str
3592     {
3593         \pgfnodealias
3594         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
3595         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
3596     }
3597 }

```

The blocks

The code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The following command will be linked to `\Block` in the environments of `nicematrix`. We define it with `\NewDocumentCommand` of `xparse` because it has an optional argument between `<` and `>` (for TeX instructions put before the math mode of the label)

It’s mandatory to use an expandable command (probably because of the first optional argument ?).

```

3598 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
3599 { \@@_Block_i #2 \q_stop { #1 } { #3 } { #4 } }

```

The first mandatory argument of `\@@_Block:` has a special syntax. It must be of the form i - j where i and j are the size (in rows and columns) of the block.

```

3600 \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 `#5` is the label of the block.

```

3601 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
3602 {
3603     \tl_set:Nx \l_tmpa_tl
3604     {
3605         { \int_use:N \c@iRow }
3606         { \int_use:N \c@jCol }
3607         { \int_eval:n { \c@iRow + #1 - 1 } }
3608         { \int_eval:n { \c@jCol + #2 - 1 } }
3609     }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components surrounded by brackets:

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

We store this information in the sequence `\g_@@_pos_of_blocks_seq`.

```

3610 \seq_gput_left:NV \g_@@_pos_of_blocks_seq \l_tmpa_tl

```

We also store a complete description of the block in the sequence `\g_@@_blocks_seq`. Of course, the sequences `\g_@@_pos_of_blocks_seq` and `\g_@@_blocks_seq` are redundant, but it's for efficiency. In `\g_@@_blocks_seq`, each block is represented by an “objet” with six components: `{imin}{jmin}{imax}{jmax}{options}{contents}`.

```

3611 \seq_gput_left:Nx \g_@@_blocks_seq
3612 {
3613   \l_tmpa_tl
3614   { #3 }
3615   \exp_not:n { { #4 \@@_math_toggle_token: #5 \@@_math_toggle_token: } }
3616 }
3617 }
```

The key `tikz` is for Tikz options used when the PGF node of the block is created (the “normal” block node and not the “short” one nor the “medium” one). **In fact, as for now, it is *not* documented.** Is it really a good idea to provide such a key?

```

3618 \keys_define:nn { NiceMatrix / Block }
3619 {
3620   tikz .tl_set:N = \l_@@_tikz_tl ,
3621   tikz .value_required:n = true ,
3622 }
```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construction of the array.

```

3623 \cs_new_protected:Npn \@@_draw_blocks:
3624 { \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iii:nnnnnn ##1 } }
3625 \cs_new_protected:Npn \@@_Block_iii:nnnnnn #1 #2 #3 #4 #5 #6
3626 {
```

The group is for the keys (even if, as for now, there is only one key, called `tikz` and not documented).

```

3627 \group_begin:
3628 \keys_set:nn { NiceMatrix / Block } { #5 }
3629 \bool_lazy_or:nnTF
3630 { \int_compare_p:nNn { #3 } > \g_@@_row_total_int }
3631 { \int_compare_p:nNn { #4 } > \c@jCol }
3632 { \msg_error:nnnn { nicematrix } { Block-too-large } { #1 } { #2 } }
3633 {
```

We put the contents of the cell in the box `\l_@@_cell_box` because we want the command `\rotate` used in the content to be able to rotate the box.

```

3634 \hbox_set:Nn \l_@@_cell_box { #6 }
```

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.

```

3635 \pgfpicture
3636 \pgfrememberpicturepositiononpagetrue
3637 \pgf@relevantforpicturesizefalse
3638 \@@_qpoint:n { row - #1 }
3639 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3640 \@@_qpoint:n { col - #2 }
3641 \dim_set_eq:NN \l_tmpb_dim \pgf@x
3642 \@@_qpoint:n { row - \@@_succ:n { #3 } }
3643 \dim_set_eq:NN \l_tmpc_dim \pgf@y
3644 \@@_qpoint:n { col - \@@_succ:n { #4 } }
3645 \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 as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```

3646 \begin { pgfscope }
3647 \exp_args:Nx \pgfset { \l_@@_tikz_tl }
3648 \@@_pgf_rect_node:nnnnn
3649 { \@@_env: - #1 - #2 - block }
3650 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
3651 \end { pgfscope }

```

We construct the short node.

```

3652 \dim_set_eq:NN \l_tmpb_dim \c_max_dim
3653 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3654 {

```

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.

```

3655 \cs_if_exist:cT
3656 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
3657 {
3658 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
3659 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
3660 }
3661 }

```

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.

```

3662 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
3663 {
3664 \@@_qpoint:n { col - #2 }
3665 \dim_set_eq:NN \l_tmpb_dim \pgf@x
3666 }
3667 \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
3668 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3669 {
3670 \cs_if_exist:cT
3671 { pgf @ sh @ ns @ \@@_env: - ##1 - #4 }
3672 {
3673 \pgfpointanchor { \@@_env: - ##1 - #4 } { east }
3674 \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
3675 }
3676 }
3677 \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
3678 {
3679 \@@_qpoint:n { col - \@@_succ:n { #4 } }
3680 \dim_set_eq:NN \l_tmpd_dim \pgf@x
3681 }
3682 \@@_pgf_rect_node:nnnnn
3683 { \@@_env: - #1 - #2 - block - short }
3684 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_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 as arguments the name of the node and two PGF points.

```

3685     \bool_if:NT \l_@@_medium_nodes_bool
3686     {
3687         \@@_pgf_rect_node:nnn
3688         { \@@_env: - #1 - #2 - block - medium }
3689         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
3690         { \pgfpointanchor { \@@_env: - #3 - #4 - medium } { south-east } }
3691     }

```

Now, we will put the label of the block.

```

3692     \int_compare:nNnTF { #1 } = { #3 }
3693     {

```

We take into account the case of a block of one row in the “first row” or the “end row”.

```

3694         \int_compare:nNnTF { #1 } = 0
3695         { \l_@@_code_for_first_row_tl }
3696         {
3697             \int_compare:nNnT { #1 } = \l_@@_last_row_int
3698             \l_@@_code_for_last_row_tl
3699         }

```

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`.

```

3700         \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

3701         \@@_qpoint:n { #1 - #2 - block - short }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

3702         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
3703         \pgfnode { rectangle } { base }
3704         { \box_use_drop:N \l_@@_cell_box } { } { }
3705     }

```

If the number of rows is different of 1, we put the label of the block in the center of the (short) node (the label of the block has been composed in `\l_@@_cell_box`).

```

3706     {
3707         \pgftransformshift { \@@_qpoint:n { #1 - #2 - block - short } }
3708         \pgfnode { rectangle } { center }
3709         { \box_use_drop:N \l_@@_cell_box } { } { }
3710     }
3711     \endpgfpicture
3712 }
3713 \group_end:
3714 }

```

How to draw the dotted lines transparently

```

3715 \cs_set_protected:Npn \@@_renew_matrix:
3716 {
3717     \RenewDocumentEnvironment { pmatrix } { } {
3718         { \pNiceMatrix }
3719         { \endpNiceMatrix }
3720     }
3721     \RenewDocumentEnvironment { vmatrix } { } {
3722         { \vNiceMatrix }
3723         { \endvNiceMatrix }
3724     }
3725     \RenewDocumentEnvironment { Vmatrix } { } {
3726         { \VNiceMatrix }
3727         { \endVNiceMatrix }
3728     }
3729     \RenewDocumentEnvironment { bmatrix } { } {

```

```

3727     { \bNiceMatrix }
3728     { \endbNiceMatrix }
3729 \RenewDocumentEnvironment { Bmatrix } { }
3730     { \BNiceMatrix }
3731     { \endBNiceMatrix }
3732 }

```

Automatic arrays

```

3733 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
3734 {
3735     \int_set:Nn \l_@@_nb_rows_int { #1 }
3736     \int_set:Nn \l_@@_nb_cols_int { #2 }
3737 }
3738 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
3739 {
3740     \int_zero_new:N \l_@@_nb_rows_int
3741     \int_zero_new:N \l_@@_nb_cols_int
3742     \@@_set_size:n #4 \q_stop
3743     \begin { NiceArrayWithDelims } { #1 } { #2 }
3744         { * { \l_@@_nb_cols_int } { C } } [ #3 , #5 , #7 ]
3745     \int_compare:nNnT \l_@@_first_row_int = 0
3746     {
3747         \int_compare:nNnT \l_@@_first_col_int = 0 { & }
3748         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
3749         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
3750     }
3751     \prg_replicate:nn \l_@@_nb_rows_int
3752     {
3753         \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

You put { } before #6 to avoid a hasty expansion of an eventual `\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`).

```

3754         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
3755         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
3756     }
3757     \int_compare:nNnT \l_@@_last_row_int > { -2 }
3758     {
3759         \int_compare:nNnT \l_@@_first_col_int = 0 { & }
3760         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
3761         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
3762     }
3763     \end { NiceArrayWithDelims }
3764 }
3765 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
3766 {
3767     \cs_set_protected:cpn { #1 AutoNiceMatrix }
3768     {
3769         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
3770         \AutoNiceMatrixWithDelims { #2 } { #3 }
3771     }
3772 }
3773 \@@_define_com:nnn p ( )
3774 \@@_define_com:nnn b [ ]
3775 \@@_define_com:nnn v | |
3776 \@@_define_com:nnn V \l \l
3777 \@@_define_com:nnn B \{ \}

```

We define also an command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

3778 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
3779 {

```

```

3780 \group_begin:
3781   \bool_set_true:N \l_@@_NiceArray_bool
3782   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
3783 \group_end:
3784 }

```

The redefinition of the command `\dotfill`

```

3785 \cs_set_eq:NN \@@_dotfill \dotfill
3786 \cs_new_protected:Npn \@@_dotfill:
3787 {

```

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}`).

```

3788   \@@_dotfill
3789   \bool_if:NT \l_@@_NiceTabular_bool
3790     { \group_insert_after:N \@@_dotfill_ii: }
3791     { \group_insert_after:N \@@_dotfill_i: }
3792 }
3793 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
3794 \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`.

```

3795 \cs_new_protected:Npn \@@_dotfill_iii:
3796 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim { \@@_dotfill }

```

The command `\diagbox`

```

3797 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
3798 {
3799   \tl_gput_right:Nx \g_@@_internal_code_after_tl
3800   {
3801     \@@_actually_diagbox:nnnn
3802     { \int_use:N \c_iRow } { \int_use:N \c_jCol } { #1 } { #2 }
3803   }
3804 }

```

The two arguments of `\@@_actually_diagbox:nn` are the number of row and the number of column of the cell to slash. The two other are the elements to draw below and above the diagonal line.

```

3805 \cs_new_protected:Npn \@@_actually_diagbox:nnnn #1 #2 #3 #4
3806 {
3807   \pgfpicture
3808   \pgf@relevantforpicturesizefalse
3809   \pgfrememberpicturepositiononpagetrue
3810   \@@_qpoint:n { row - #1 }
3811   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3812   \@@_qpoint:n { col - #2 }
3813   \dim_set_eq:NN \l_tmpb_dim \pgf@x
3814   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
3815   \@@_qpoint:n { row - \@@_succ:n { #1 } }
3816   \dim_set_eq:NN \l_tmpc_dim \pgf@y
3817   \@@_qpoint:n { col - \@@_succ:n { #2 } }
3818   \dim_set_eq:NN \l_tmpd_dim \pgf@x
3819   \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
3820   {

```

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.

```

3821   \CT@arc@
3822   \pgfsetroundcap
3823   \pgfusepathqstroke
3824 }

```

```

3825 \pgfset { inner~sep = 1 pt }
3826 \pgfscope
3827 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
3828 \pgfnode { rectangle } { south~west }
3829 { \@@_math_toggle_token: #3 \@@_math_toggle_token: } { } { }
3830 \endpgfscope
3831 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
3832 \pgfnode { rectangle } { north~east }
3833 { \@@_math_toggle_token: #4 \@@_math_toggle_token: } { } { }
3834 \endpgfpicture
3835 }

```

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. 67.

The command `\CodeAfter` catches everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

3836 \cs_new_protected:Npn \@@_CodeAfter:n #1 \end
3837 {
3838   \tl_gput_right:Nn \g_@@_code_after_tl { #1 }
3839   \@@_CodeAfter_i:n
3840 }

```

We catch the argument of the command `\end` (in `#1`).

```

3841 \cs_new_protected:Npn \@@_CodeAfter_i:n #1
3842 {

```

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.

```

3843   \str_if_eq:eeTF \@@_currenvir { #1 }
3844   { \end { #1 } }

```

If this is not the `\end` we are looking for, we put those tokens in `\g_@@_code_after_tl` and we go on searching for the next command `\end` with a recursive call to the command `\@@_CodeAfter:n`.

```

3845   {
3846     \tl_gput_right:Nn \g_@@_code_after_tl { \end { #1 } }
3847     \@@_CodeAfter:n
3848   }
3849 }

```

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.

```

3850 \keys_define:nn { NiceMatrix / Package }
3851 {
3852   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
3853   renew-dots .value_forbidden:n = true ,
3854   renew-matrix .code:n = \@@_renew_matrix: ,
3855   renew-matrix .value_forbidden:n = true ,
3856   transparent .meta:n = { renew-dots , renew-matrix } ,
3857   transparent .value_forbidden:n = true,
3858 }
3859 \ProcessKeysOptions { NiceMatrix / Package }

```

Error messages of the package

The following command converts all the elements of a sequence (which are token lists) into strings.

```

3860 \cs_new_protected:Npn \@@_convert_to_str_seq:N #1
3861 {
3862   \seq_clear:N \l_tmpa_seq
3863   \seq_map_inline:Nn #1
3864   {
3865     \seq_put_left:Nx \l_tmpa_seq { \tl_to_str:n { ##1 } }
3866   }
3867   \seq_set_eq:NN #1 \l_tmpa_seq
3868 }

```

The following command creates a sequence of strings (str) from a clist.

```

3869 \cs_new_protected:Npn \@@_set_seq_of_str_from_clist:Nn #1 #2
3870 {
3871   \seq_set_from_clist:Nn #1 { #2 }
3872   \@@_convert_to_str_seq:N #1
3873 }
3874 \@@_set_seq_of_str_from_clist:Nn \c_@@_types_of_matrix_seq
3875 {
3876   NiceMatrix ,
3877   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
3878 }

```

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`.

```

3879 \cs_new_protected:Npn \@@_error_too_much_cols:
3880 {
3881   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
3882   {
3883     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
3884     { \@@_fatal:n { too-much-cols-for-matrix } }
3885     {
3886       \bool_if:NF \l_@@_last_col_without_value_bool
3887       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
3888     }
3889   }
3890   { \@@_fatal:n { too-much-cols-for-array } }
3891 }

```

The following command must *not* be protected since it's used in an error message.

```

3892 \cs_new:Npn \@@_message_hdotsfor:
3893 {
3894   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
3895   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is~incorrect.}
3896 }
3897 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
3898 {
3899   You-try-to-use-more-columns-than-allowed-by-your~
3900   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal-number-of~
3901   columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the-potential~
3902   exterior~ones).~This-error-is-fatal.
3903 }
3904 \@@_msg_new:nn { too-much-cols-for-matrix }
3905 {
3906   You-try-to-use-more-columns-than-allowed-by-your~
3907   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
3908   number-of-columns-for-a-matrix-is-fixed-by-the-LaTeX-counter~
3909   'MaxMatrixCols'.~Its-actual-value-is~\int_use:N \cMaxMatrixCols.~

```

```

3910     This-error-is-fatal.
3911 }

```

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`.

```

3912 \@@_msg_new:nn { too-much-cols-for-array }
3913 {
3914     You-try-to-use-more-columns-than-allowed-by-your-
3915     \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is-
3916     \int_eval:n { \c@jCol - 1 }~(plus-the-potential-exterior-ones).~
3917     This-error-is-fatal.
3918 }
3919 \@@_msg_new:nn { in-first-col }
3920 {
3921     You-can't-use-the-command~#1 in-the-first-column-(number~0)-of-the-array.\\
3922     If-you-go-on,~this-command-will-be-ignored.
3923 }
3924 \@@_msg_new:nn { in-last-col }
3925 {
3926     You-can't-use-the-command~#1 in-the-last-column-(exterior)-of-the-array.\\
3927     If-you-go-on,~this-command-will-be-ignored.
3928 }
3929 \@@_msg_new:nn { in-first-row }
3930 {
3931     You-can't-use-the-command~#1 in-the-first-row-(number~0)-of-the-array.\\
3932     If-you-go-on,~this-command-will-be-ignored.
3933 }
3934 \@@_msg_new:nn { in-last-row }
3935 {
3936     You-can't-use-the-command~#1 in-the-last-row-(exterior)-of-the-array.\\
3937     If-you-go-on,~this-command-will-be-ignored.
3938 }
3939 \@@_msg_new:nn { option-S-without-siunitx }
3940 {
3941     You-can't-use-the-option-'S'-in-your-environment-\@@_full_name_env:
3942     because-you-have-not-loaded-siunitx.\\
3943     If-you-go-on,~this-option-will-be-ignored.
3944 }
3945 \@@_msg_new:nn { bad-option-for-line-style }
3946 {
3947     Since-you-haven't-loaded-Tikz,~the-only-value-you-can-give-to-'line-style'-
3948     is-'standard'.~If-you-go-on,~this-option-will-be-ignored.
3949 }
3950 \@@_msg_new:nn { Unknown-option-for~xdots }
3951 {
3952     As-for-now-there-is-only-three-options-available-here:~'color',~'line-style'-
3953     and~'shorten'~(and-you-try-to-use~'\l_keys_key_tl').~If-you-go-on,~
3954     this-option-will-be-ignored.
3955 }
3956 \@@_msg_new:nn { ampersand-in-light-syntax }
3957 {
3958     You-can't-use-an-ampersand-(\token_to_str &)~to-separate-columns-because
3959     ~you-have-used-the-option-'light-syntax'.~This-error-is-fatal.
3960 }
3961 \@@_msg_new:nn { double-backslash-in-light-syntax }
3962 {
3963     You-can't-use~\token_to_str:N \\~to-separate-rows-because-you-have-used-
3964     the-option-'light-syntax'.~You-must-use-the-character~'\l_@@_end_of_row_tl'-
3965     (set-by-the-option-'end-of-row').~This-error-is-fatal.
3966 }

```

```

3967 \@@_msg_new:nn { standard-cline-in-document }
3968 {
3969     The~key~'standard-cline'~is~available~only~in~the~preamble.\\
3970     If~you~go~on~this~command~will~be~ignored.
3971 }
3972 \@@_msg_new:nn { bad-value-for-baseline }
3973 {
3974     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
3975     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
3976     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
3977     If~you~go~on,~a~value~of~1~will~be~used.
3978 }
3979 \@@_msg_new:nn { empty-environment }
3980 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }
3981 \@@_msg_new:nn { unknown-cell-for-line-in-code-after }
3982 {
3983     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code-after'~
3984     can't~be~executed~because~a~cell~doesn't~exist.\\
3985     If~you~go~on~this~command~will~be~ignored.
3986 }
3987 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
3988 {
3989     In~the~\@@_full_name_env:,~you~must~use~the~option~
3990     'last-col'~without~value.\\
3991     However,~you~can~go~on~for~this~time~
3992     (the~value~'\l_keys_value_tl'~will~be~ignored).
3993 }
3994 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
3995 {
3996     In~\NiceMatrixoptions,~you~must~use~the~option~
3997     'last-col'~without~value.\\
3998     However,~you~can~go~on~for~this~time~
3999     (the~value~'\l_keys_value_tl'~will~be~ignored).
4000 }
4001 \@@_msg_new:nn { Block-too-large }
4002 {
4003     You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
4004     too~small~for~that~block. \\
4005     If~you~go~on,~this~command~will~be~ignored.
4006 }
4007 \@@_msg_new:nn { Wrong-last-row }
4008 {
4009     You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
4010     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
4011     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
4012     last~row.~You~can~avoid~this~problem~by~using~'last-row'~
4013     without~value~(more~compilations~might~be~necessary).
4014 }
4015 \@@_msg_new:nn { Yet-in-env }
4016 {
4017     Environments~\{NiceArray\}~(or~\{NiceMatrix\},~etc.)~can't~be~nested.\\
4018     This~error~is~fatal.
4019 }
4020 \@@_msg_new:nn { Outside-math-mode }
4021 {
4022     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
4023     (and~not~in~\token_to_str:N \vcenter).\\
4024     This~error~is~fatal.
4025 }

```

```

4026 \@_msg_new:nn { Bad-value-for-letter-for-dotted-lines }
4027 {
4028   The-value-of-key~'\tl_use:N\l_keys_key_tl'~must-be-of-length~1.\
4029   If-you-go-on,~it~will~be~ignored.
4030 }
4031 \@_msg_new:nnn { Unknown-key-for-NiceMatrixOptions }
4032 {
4033   The-key~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~command~
4034   \token_to_str:N \NiceMatrixOptions. \
4035   If-you-go-on,~it~will~be~ignored. \
4036   For-a-list-of-the~*principal*~available-keys,~type~H~<return>.
4037 }
4038 {
4039   The~available~options~are~(in~alphabetic~order):~
4040   allow-duplicate-names,~
4041   code-for-first-col,~
4042   cell-space-bottom-limit,~
4043   cell-space-top-limit,~
4044   code-for-first-row,~
4045   code-for-last-col,~
4046   code-for-last-row,~
4047   create-extra-nodes,~
4048   create-medium-nodes,~
4049   create-large-nodes,~
4050   end-of-row,~
4051   first-col,~
4052   first-row,~
4053   hlines,~
4054   hvlines,~
4055   hvlines-except-corners,~
4056   last-col,~
4057   last-row,~
4058   left-margin,~
4059   letter-for-dotted-lines,~
4060   light-syntax,~
4061   nullify-dots,~
4062   renew-dots,~
4063   renew-matrix,~
4064   right-margin,~
4065   small,~
4066   transparent,~
4067   vlines,~
4068   xdots/color,~
4069   xdots/shorten-and~
4070   xdots/line-style.
4071 }
4072 \@_msg_new:nnn { Unknown-option-for-NiceArray }
4073 {
4074   The-option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~environment~
4075   \{NiceArray\}. \
4076   If-you-go-on,~it~will~be~ignored. \
4077   For-a-list-of-the~*principal*~available~options,~type~H~<return>.
4078 }
4079 {
4080   The~available~options~are~(in~alphabetic~order):~
4081   b,~
4082   baseline,~
4083   c,~
4084   cell-space-bottom-limit,~
4085   cell-space-top-limit,~
4086   code-after,~
4087   code-for-first-col,~
4088   code-for-first-row,~

```

```

4089     code-for-last-col,~
4090     code-for-last-row,~
4091     columns-width,~
4092     create-extra-nodes,~
4093     create-medium-nodes,~
4094     create-large-nodes,~
4095     extra-left-margin,~
4096     extra-right-margin,~
4097     first-col,~
4098     first-row,~
4099     hlines,~
4100     hvlines,~
4101     last-col,~
4102     last-row,~
4103     left-margin,~
4104     light-syntax,~
4105     name,~
4106     nullify-dots,~
4107     renew-dots,~
4108     right-margin,~
4109     rules/color,~
4110     rules/width,~
4111     small,~
4112     t,~
4113     vlines,~
4114     xdots/color,~
4115     xdots/shorten~and~
4116     xdots/line-style.
4117 }

```

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 options `t`, `c` and `b`).

```

4118 \@@_msg_new:nnn { Unknown-option-for-NiceMatrix }
4119 {
4120   The~option~'\tl_use:N\l_keys_key_tl'~is-unknown-for-the~
4121   \@@_full_name_env:. \\\
4122   If~you~go~on,~it~will~be~ignored. \\\
4123   For~a~list~of~the~*principal*~available~options,~type~H~<return>.
4124 }
4125 {
4126   The~available~options~are~(in~alphabetic~order):~
4127   b,~
4128   baseline,~
4129   c,~
4130   cell-space-bottom-limit,~
4131   cell-space-top-limit,~
4132   code-after,~
4133   code-for-first-col,~
4134   code-for-first-row,~
4135   code-for-last-col,~
4136   code-for-last-row,~
4137   columns-width,~
4138   create-extra-nodes,~
4139   create-medium-nodes,~
4140   create-large-nodes,~
4141   extra-left-margin,~
4142   extra-right-margin,~
4143   first-col,~
4144   first-row,~
4145   hlines,~
4146   hvlines,~
4147   l~(=L),~
4148   last-col,~

```

```

4149     last-row,~
4150     left-margin,~
4151     light-syntax,~
4152     name,~
4153     nullify-dots,~
4154     r~(=R),~
4155     renew-dots,~
4156     right-margin,~
4157     rules/color,~
4158     rules/width,~
4159     S,~
4160     small,~
4161     t,~
4162     vlines,~
4163     xdots/color,~
4164     xdots/shorten~and~
4165     xdots/line-style.
4166 }

4167 \@@_msg_new:nnn { Unknown~option~for~NiceTabular }
4168 {
4169     The~option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~environment~
4170     \{NiceTabular\}. \\
4171     If~you~go~on,~it~will~be~ignored. \\
4172     For~a~list~of~the~*principal*~available~options,~type~H~<return>.
4173 }
4174 {
4175     The~available~options~are~(in~alphabetic~order):~
4176     b,~
4177     baseline,~
4178     c,~
4179     cell-space-bottom-limit,~
4180     cell-space-top-limit,~
4181     code-after,~
4182     code-for-first-col,~
4183     code-for-first-row,~
4184     code-for-last-col,~
4185     code-for-last-row,~
4186     columns-width,~
4187     create-extra-nodes,~
4188     create-medium-nodes,~
4189     create-large-nodes,~
4190     extra-left-margin,~
4191     extra-right-margin,~
4192     first-col,~
4193     first-row,~
4194     hlines,~
4195     hvlines,~
4196     last-col,~
4197     last-row,~
4198     left-margin,~
4199     light-syntax,~
4200     name,~
4201     nullify-dots,~
4202     renew-dots,~
4203     right-margin,~
4204     rules/color,~
4205     rules/width,~
4206     t,~
4207     vlines,~
4208     xdots/color,~
4209     xdots/shorten~and~
4210     xdots/line-style.
4211 }

```

```

4212 \@@_msg_new:nnn { Duplicate-name }
4213 {
4214   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
4215   the~same~environment~name~twice.~You~can~go~on,~but,~
4216   maybe,~you~will~have~incorrect~results~especially~
4217   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
4218   message~again,~use~the~option~'allow-duplicate-names'.\\
4219   For~a~list~of~the~names~already~used,~type~H~<return>. \\
4220 }
4221 {
4222   The~names~already~defined~in~this~document~are:~
4223   \seq_use:Nnnn \g_@@_names_seq { ,~ } { ,~ } { ~and~ }.
4224 }
4225 \@@_msg_new:nn { Option-auto-for-columns-width }
4226 {
4227   You~can't~give~the~value~'auto'~to~the~option~'columns-width'~here.~
4228   If~you~go~on,~the~option~will~be~ignored.
4229 }
4230 \@@_msg_new:nn { Zero-row }
4231 {
4232   There~is~a~problem.~Maybe~you~have~used~l,~c~and~r~instead~of~L,~C~
4233   and~R~in~the~preamble~of~your~environment. \\
4234   This~error~is~fatal.
4235 }

```

16 History

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 StackExchange⁴¹, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁴²

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`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

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`.

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`).

⁴¹cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁴²Before this version, there was an error when using `nicematrix` with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by `nicematrix` because they use the options `overlay` and `remember picture`.

Changes between version 2.2 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 StackExchange⁴³, optimization of Tikz externalization is disabled in the environments of `nicematrix` when the class `standalone` or the package `standalone` is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

⁴³cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between version 3.5 and 3.6

LaTeX counters `iRow` and `jCol` available in the cells of the array.

Addition of `\normalbaselines` before the construction of the array: in environments like `{align}` of `amsmath` the value of `\baselineskip` is changed and if the options `first-row` and `last-row` were used in an environment of `nicematrix`, the position of the delimiters was wrong.

A warning is written in the `.log` file if an obsolete environment is used.

There is no longer artificial errors `Duplicate~name` in the environments of `amsmath`.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.

New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programming for the command `\Block` when the block has only one row. With this programming, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with `\multicolumn`. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands `\NiceMatrixLastEnv` and `\OnlyMainNiceMatrix`.

New options `create-medium-nodes` and `create-large-nodes`.

Changes between version 3.9 and 3.10

New option `light-syntax` (and `end-of-row`).

New option `dotted-lines-margin` for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

Correction of a bug linked to `first-row` and `last-row`.

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -`block` and, if the creation of the “medium nodes” is required, a node $i-j$ -`block-medium` is created.

If the user try 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 command `\CodeAfter` (in the environments of `nicematrix`).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`

Commands to color cells, row 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`.

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:	
\@@_Block:	810, 3598
\@@_Block_i	3599, 3600
\@@_Block_ii:nnnnn	3600, 3601
\@@_Block_iii:nnnnnn	3624, 3625
\@@_Cdots	801, 819, 2475
\g_@@_Cdots_lines_tl	890, 1800
\@@_Cell: 167, 547, 737, 832, 833, 834, 842, 856	
\@@_CodeAfter:n	814, 3836, 3847
\@@_CodeAfter_i:n	3839, 3841
\@@_Ddots	803, 821, 2505
\g_@@_Ddots_lines_tl	893, 1798
\g_@@_HVDotsfor_lines_tl	895, 1796, 2588, 2664, 3894
\@@_Hdotsfor:	807, 824, 2577
\@@_Hdotsfor:nnnn	2590, 2602
\@@_Hdotsfor_i	2580, 2586
\@@_Hspace:	806, 2550
\@@_Iddots	804, 822, 2527
\g_@@_Iddots_lines_tl	894, 1799
\@@_Ldots	800, 818, 823, 2460
\g_@@_Ldots_lines_tl	891, 1801
\l_@@_NiceArray_bool	184, 924, 1035, 1053, 1065, 1120, 1569, 3012, 3013, 3032, 3033, 3063, 3064, 3122, 3123, 3781
\g_@@_NiceMatrixBlock_int	180, 3389, 3394, 3397, 3408
\l_@@_NiceTabular_bool	128, 185, 554, 764, 906, 979, 981, 1054, 1066, 1074, 1191, 1195, 1649, 2019, 2029, 2063, 2071, 2735, 3328, 3335, 3789
\@@_OnlyMainNiceMatrix:n	812, 2978, 2997
\@@_OnlyMainNiceMatrix_i:n	2981, 2988, 2991
\@@_Vdots	802, 820, 2490
\g_@@_Vdots_lines_tl	892, 1797
\@@_Vdotsfor:	808, 2662
\@@_Vdotsfor:nnnn	2666, 2677
\@@_actually_diagbox:nnnn	3801, 3805
\@@_actually_draw_Cdots:	2052, 2056
\@@_actually_draw_Ddots:	2179, 2183
\@@_actually_draw_Iddots:	2230, 2234
\@@_actually_draw_Ldots:	2008, 2012, 2653
\@@_actually_draw_Vdots:	2106, 2110, 2728
\@@_adapt_S_column:	142, 157, 905
\@@_after_array:	1213, 1653
\@@_analyze_end:Nn	1294, 1339
\l_@@_argspec_tl	2458, 2459, 2460, 2475, 2490, 2505, 2527, 2584, 2585, 2586, 2660, 2661, 2662, 2758, 2759, 2760
\@@_array:	666, 1295, 1322
\l_@@_auto_columns_width_bool	306, 417, 1403, 1407, 3384
\l_@@_baseline_str	296, 297, 410, 411, 412, 413, 677, 1122, 1138, 1141, 1142, 1143, 1220, 1226, 1231
\@@_begin_of_NiceMatrix:nn	1622, 1626
\@@_begin_of_row:	551, 575, 1489
\l_@@_block_auto_columns_width_bool	919, 1408, 3377, 3382, 3392, 3402
\g_@@_blocks_seq	220, 922, 1759, 3611, 3624
\c_@@_booktabs_loaded_bool	21, 26, 747
\l_@@_cell_box	553, 599, 601, 607, 615, 616, 617, 618, 620, 623, 625, 627, 644, 749, 841, 849, 855, 864, 990, 992, 1490, 1512, 1515, 1517, 1532, 1553, 1557, 2743, 2746, 2750, 3634, 3704, 3709, 3796
\l_@@_cell_space_bottom_limit_dim	285, 350, 618
\l_@@_cell_space_top_limit_dim	284, 348, 616
\@@_cellcolor	973, 2886, 2971, 2972
\g_@@_cells_seq	1333, 1334, 1335, 1337
\@@_chessboardcolors	978, 2964
\@@_cline	111, 799
\@@_cline_i:nn	112, 113, 121, 124
\@@_cline_i:w	113, 114
\g_@@_code_after_tl	203, 430, 1316, 1776, 1777, 3838, 3846
\l_@@_code_before_bool	209, 407, 684, 930, 1353, 1367, 1385, 1416, 1442, 1465, 1699
\l_@@_code_before_tl	406, 980
\l_@@_code_for_first_col_tl	362, 1501
\l_@@_code_for_first_row_tl	366, 563, 3695
\l_@@_code_for_last_col_tl	364, 1541
\l_@@_code_for_last_row_tl	368, 570, 3698
\g_@@_col_total_int	552, 830, 1111, 1435, 1436, 1467, 1472, 1477, 1478, 1531, 1657, 1660, 1665, 1672, 1716, 2143, 2723, 3425, 3435, 3469, 3556
\c_@@_colortbl_loaded_bool	75, 79, 766
\@@_columncolor	977, 2850
\l_@@_columns_width_dim	181, 418, 484, 1404, 1410, 3390, 3396
\g_@@_com_or_env_str	195, 196, 199
\@@_computations_for_large_nodes:	3496, 3509, 3514
\@@_computations_for_medium_nodes:	3416, 3485, 3495, 3506
\@@_compute_a_corner:nnnnnn	3210, 3211, 3212, 3213, 3216
\@@_convert_to_str_seq:N	3860, 3872
\@@_create_col_nodes:	1298, 1326, 1345
\@@_create_large_nodes:	1736, 3490
\@@_create_medium_and_large_nodes:	1733, 3501
\@@_create_medium_nodes:	1734, 3480
\@@_create_nodes:	3487, 3498, 3508, 3511, 3552
\@@_create_row_node:	680, 712, 748
\@@_cut_on_hyphen:w	2809, 2831, 2832, 2865, 2866, 2895, 2926, 2937
\g_@@_ddots_int	1723, 2203, 2204
\@@_define_columntype:nn	732, 835, 836, 837
\@@_define_com:nnn	3765, 3773, 3774, 3775, 3776, 3777
\@@_define_env:n	1615, 1639, 1640, 1641, 1642, 1643, 1644

<code>\g_@@_delta_x_one_dim</code>	1725, 2206, 2216	<code>\@@_error:n</code>	12,
<code>\g_@@_delta_x_two_dim</code>	1727, 2257, 2267		326, 335, 472, 483, 492, 495, 515, 518,
<code>\g_@@_delta_y_one_dim</code>	1726, 2208, 2216		525, 527, 533, 538, 543, 545, 1106, 1150, 1238
<code>\g_@@_delta_y_two_dim</code>	1728, 2259, 2267	<code>\@@_error:nn</code>	13,
<code>\@@_diagbox:nn</code>	815, 3797		425, 2463, 2466, 2478, 2481, 2493, 2496,
<code>\@@_dotfill</code>	3785, 3788, 3796		2509, 2510, 2515, 2516, 2531, 2532, 2537, 2538
<code>\@@_dotfill:</code>	813, 3786	<code>\@@_error:nnn</code>	14, 2785
<code>\@@_dotfill_i:</code>	3791, 3793	<code>\@@_error_too_much_cols:</code>	1072, 3879
<code>\@@_dotfill_ii:</code>	3790, 3793, 3794	<code>\@@_everycr:</code>	706, 771, 774
<code>\@@_dotfill_iii:</code>	3794, 3795	<code>\@@_everycr_i:</code>	706, 707
<code>\@@_double_int_eval:n</code>	2754, 2768, 2769	<code>\l_@@_exterior_arraycolsep_bool</code>	
<code>\g_@@_dp_ante_last_row_dim</code>	578, 782		298, 480, 1056, 1068
<code>\g_@@_dp_last_row_dim</code>		<code>\l_@@_extra_left_margin_dim</code>	
	578, 579, 785, 786, 991, 992, 1179		312, 392, 1085, 1520
<code>\g_@@_dp_row_zero_dim</code>		<code>\l_@@_extra_right_margin_dim</code>	
	598, 599, 776, 777, 1133, 1162, 1172		313, 393, 1097, 1561, 2146
<code>\@@_draw_Cdots:nnn</code>	2038	<code>\@@_extract_coords_values:</code>	3577, 3584
<code>\@@_draw_Ddots:nnn</code>	2171	<code>\@@_fatal:n</code>	15, 189, 909,
<code>\@@_draw_Iddots:nnn</code>	2222		1303, 1307, 1309, 1342, 1348, 3884, 3887, 3890
<code>\@@_draw_Ldots:nnn</code>	1994	<code>\@@_fatal:nn</code>	16
<code>\@@_draw_Vdots:nnn</code>	2092	<code>\l_@@_final_i_int</code>	
<code>\@@_draw_blocks:</code>	1759, 3623		1739, 1813, 1818, 1821, 1846,
<code>\@@_draw_dotted_lines:</code>	1749, 1785		1854, 1858, 1867, 1875, 1957, 1988, 2030,
<code>\@@_draw_dotted_lines_i:</code>	1788, 1792		2128, 2195, 2246, 2607, 2635, 2703, 2713, 2715
<code>\@@_draw_hlines:</code>	1753, 3024, 3055	<code>\l_@@_final_j_int</code>	
<code>\@@_draw_hvlines:</code>	1751, 3045		1740, 1814, 1819, 1826, 1831, 1837, 1847,
<code>\@@_draw_hvlines_except_corners:</code>	1756, 3207		1855, 1859, 1868, 1876, 1958, 1989, 2026,
<code>\@@_draw_hvlines_i:</code>	3050, 3053		2068, 2197, 2248, 2628, 2638, 2640, 2682, 2711
<code>\@@_draw_hvlines_ii:</code>	3051, 3058, 3214	<code>\l_@@_final_open_bool</code>	1742, 1820,
<code>\@@_draw_line:</code>	2036,		1824, 1827, 1834, 1840, 1844, 1860, 2024,
	2090, 2169, 2220, 2271, 2273, 2807, 3345, 3375		2066, 2076, 2087, 2113, 2126, 2134, 2155,
<code>\@@_draw_line_ii:nn</code>	2787, 2791		2193, 2244, 2380, 2395, 2426, 2427, 2605,
<code>\@@_draw_line_iii:nn</code>	2794, 2798		2629, 2641, 2680, 2704, 2716, 2777, 3311, 3350
<code>\@@_draw_non_standard_dotted_line:</code> . .		<code>\@@_find_extremities_of_line:nnnn</code> . . .	
	2279, 2281		1808, 1998, 2042, 2097, 2175, 2226
<code>\@@_draw_non_standard_dotted_line:n</code> . .		<code>\l_@@_first_col_int</code>	103, 112,
	2284, 2287		225, 226, 358, 531, 551, 1048, 1115, 1349,
<code>\@@_draw_standard_dotted_line:</code> .	2278, 2307		1365, 1710, 2822, 2873, 2905, 2934, 2980,
<code>\@@_draw_standard_dotted_line_i:</code>	2372, 2376		3425, 3435, 3469, 3517, 3556, 3747, 3753, 3759
<code>\@@_draw_vlines:</code>	1754, 2998, 3056	<code>\l_@@_first_row_int</code>	
<code>\g_@@_empty_cell_bool</code>	216,		223, 224, 359, 535, 828,
	622, 629, 2473, 2488, 2503, 2525, 2547, 2552		1130, 1146, 1159, 1170, 1234, 1708, 3418,
<code>\l_@@_empty_corner_cells_seq</code>			3432, 3459, 3516, 3554, 3653, 3668, 3745, 3975
	3079, 3085, 3092, 3136, 3142, 3149, 3209, 3271	<code>\@@_full_name_env:</code>	197, 3900,
<code>\@@_end_Cell:</code>			3907, 3915, 3941, 3980, 3989, 4010, 4022, 4121
	169, 611, 742, 832, 833, 834, 846, 860	<code>\@@_hdottedline:</code>	805, 3296
<code>\l_@@_end_of_row_tl</code>		<code>\@@_hdottedline:n</code>	3304, 3308
	314, 315, 356, 1318, 1319, 3964	<code>\@@_hdottedline_i:</code>	3299, 3301
<code>\c_@@_endpgfortikzpicture_tl</code>		<code>\@@_hdottedline_i:n</code>	3313, 3317
	32, 36, 1789, 2795, 3314	<code>\l_@@_hlines_bool</code> . .	301, 370, 376, 713, 1753
<code>\@@_env:</code>	175, 179, 584, 590, 645, 651,	<code>\g_@@_ht_last_row_dim</code>	
	689, 695, 701, 944, 945, 951, 952, 959, 960,		580, 783, 784, 989, 990, 1178
	970, 1354, 1357, 1359, 1372, 1378, 1381,	<code>\g_@@_ht_row_one_dim</code>	606, 607, 780, 781
	1390, 1396, 1399, 1421, 1427, 1430, 1447,	<code>\g_@@_ht_row_zero_dim</code>	
	1453, 1459, 1467, 1472, 1478, 1770, 1866,		600, 601, 778, 779, 1133, 1162, 1173
	1934, 1976, 1987, 2616, 2634, 2691, 2709,	<code>\l_@@_hlines_bool</code> . . .	303, 374, 1750, 1951
	2780, 2782, 2801, 2804, 3226, 3244, 3262,	<code>\l_@@_hlines_except_corners_bool</code> . . .	
	3438, 3440, 3448, 3559, 3568, 3586, 3649,		304, 400, 1755, 3074, 3131
	3656, 3658, 3671, 3673, 3683, 3688, 3689, 3690	<code>\@@_i:</code>	3418, 3420,
<code>\g_@@_env_int</code>	174, 175, 177, 918,		3421, 3422, 3423, 3432, 3438, 3440, 3441,
	932, 936, 939, 948, 949, 956, 957, 1000,		3442, 3443, 3448, 3449, 3450, 3451, 3459,
	1003, 1018, 1021, 1664, 1685, 1703, 1706, 3595		3462, 3464, 3465, 3466, 3518, 3520, 3523,

3524, 3528, 3529, 3554, 3559, 3561, 3563, 3567, 3568, 3579, 3586, 3588, 3590, 3594, 3595	\l_@@_light_syntax_bool 295, 354, 1087, 1092, 1680
\g_@@_iddots_int 1724, 2254, 2255	\@@_light_syntax_i 1311, 1314
\l_@@_in_env_bool 183, 909, 910	\@@_line 1774, 2760
\l_@@_initial_i_int 1737, 1811, 1886, 1889, 1914, 1922, 1926, 1935, 1943, 1955, 1977, 2020, 2078, 2080, 2122, 2187, 2238, 2606, 2607, 2617, 2685, 2695, 2697	\@@_line_i:nn 2767, 2774
\l_@@_initial_j_int 1738, 1812, 1887, 1894, 1899, 1905, 1915, 1923, 1927, 1936, 1944, 1956, 1978, 2016, 2060, 2136, 2138, 2143, 2189, 2240, 2610, 2620, 2622, 2681, 2682, 2693	\@@_line_with_light_syntax:n ... 1325, 1329
\l_@@_initial_open_bool 1741, 1888, 1892, 1895, 1902, 1908, 1912, 1928, 2014, 2058, 2075, 2085, 2113, 2120, 2132, 2185, 2236, 2378, 2425, 2604, 2611, 2623, 2679, 2686, 2698, 2776, 3310, 3349	\@@_line_with_light_syntax_i:n 1324, 1330, 1331
\@@_instruction_of_type:nn 655, 2468, 2483, 2498, 2519, 2541	\@@_math_toggle_token: 127, 613, 1491, 1508, 1533, 1549, 3615, 3829, 3833
\l_@@_inter_dots_dim 286, 287, 1746, 2383, 2390, 2401, 2409, 2416, 2421, 2433, 2441, 3340, 3343, 3371, 3373	\g_@@_max_cell_width_dim 619, 620, 920, 1409, 3383, 3409
\g_@@_internal_code_after_tl 204, 881, 1760, 1761, 3303, 3799	\l_@@_max_delimiter_width_bool 317, 353, 1204
\@@_j: 3425, 3427, 3428, 3429, 3430, 3435, 3438, 3440, 3443, 3445, 3446, 3448, 3451, 3453, 3454, 3469, 3472, 3474, 3475, 3476, 3531, 3533, 3536, 3538, 3542, 3543, 3556, 3559, 3560, 3562, 3567, 3568, 3580, 3586, 3587, 3589, 3594, 3595	\c_@@_max_l_dim 2370, 2375
\l_@@_l_dim 2356, 2357, 2370, 2371, 2383, 2389, 2400, 2408, 2416, 2421, 2433, 2434, 2441, 2442	\l_@@_medium_nodes_bool 308, 382, 1730, 3685
\l_@@_large_nodes_bool 309, 383, 1732, 1736	\@@_message_hdotsfor: 3892, 3900, 3907, 3915
\g_@@_last_col_found_bool 233, 889, 1112, 1208, 1434, 1463, 1529, 1656, 1713	\@@_msg_new:nn 17, 3897, 3904, 3912, 3919, 3924, 3929, 3934, 3939, 3945, 3950, 3956, 3961, 3967, 3972, 3979, 3981, 3987, 3994, 4001, 4007, 4015, 4020, 4026, 4225, 4230
\l_@@_last_col_int 231, 232, 473, 506, 508, 526, 534, 544, 942, 1014, 1020, 1027, 1060, 1632, 1634, 1657, 1660, 1712, 2102, 2141, 2465, 2480, 2516, 2538, 3749, 3755, 3761, 3883, 3901	\@@_msg_new:nnn 18, 4031, 4072, 4118, 4167, 4212
\l_@@_last_col_without_value_bool 230, 505, 1658, 3886	\@@_msg_redirect_name:nn 19, 486
\l_@@_last_empty_column_int 3237, 3250, 3256, 3269	\@@_multicolumn:nnn 809, 2556
\l_@@_last_empty_row_int . 3219, 3233, 3253	\g_@@_multicolumn_cells_seq 826, 2563, 3443, 3451, 3573
\l_@@_last_row_int .. 227, 228, 360, 568, 717, 875, 940, 985, 995, 1002, 1009, 1100, 1104, 1107, 1114, 1176, 1320, 1321, 1497, 1498, 1538, 1539, 1679, 2003, 2047, 2495, 2510, 2532, 2744, 2986, 2994, 3697, 3757, 4009	\g_@@_multicolumn_sizes_seq 827, 2565, 3574
\l_@@_last_row_without_value_bool 229, 997, 1102, 1677	\g_@@_name_env_str 194, 200, 201, 903, 904, 1341, 1570, 1571, 1577, 1578, 1585, 1586, 1593, 1594, 1601, 1602, 1609, 1610, 1619, 1647, 1779, 3769, 3881
\g_@@_last_vdotted_col_int 878, 880, 886, 888	\l_@@_name_str 307, 427, 586, 589, 647, 650, 697, 700, 998, 1007, 1010, 1016, 1025, 1028, 1358, 1359, 1380, 1381, 1398, 1399, 1429, 1430, 1455, 1458, 1474, 1477, 1667, 1671, 1688, 1692, 3564, 3567, 3591, 3594
\l_@@_left_delim_dim 1033, 1037, 1042, 1285, 1518	\g_@@_names_seq 182, 424, 426, 4223
\l_@@_left_delim_tl 900, 3339	\l_@@_nb_cols_int 3736, 3741, 3744, 3748, 3754, 3760
\l_@@_left_margin_dim 310, 386, 1084, 1519, 3329, 3547	\l_@@_nb_rows_int 3735, 3740, 3751
\l_@@_letter_for_dotted_lines_str 491, 497, 498, 868, 869	\@@_newcolumnntype 723, 734, 832, 833, 834, 838, 852
	\@@_node_for_multicolumn:nn 3575, 3582
	\@@_node_for_the_cell: 626, 631, 1516, 1562
	\@@_node_position: 951, 953, 959, 961
	\l_@@_nullify_dots_bool 305, 381, 2472, 2487, 2502, 2524, 2546
	\@@_old_CT@arc@ 911, 1781
	\@@_old_arraycolsep_dim 217, 765, 1076
	\@@_old_cdots 791, 2487
	\@@_old_ddots 793, 2524
	\l_@@_old_iRow_int 205, 751, 1805
	\@@_old_ialign: 679, 787, 1758
	\@@_old_iddots 794, 2546
	\l_@@_old_jCol_int 206, 754, 1806
	\@@_old_ldots 790, 2472
	\@@_old_multicolumn 2555, 2558
	\@@_old_pgful@check@rerun 68, 72
	\@@_old_vdots 792, 2502
	\l_@@_parallelize_diags_bool 299, 300, 378, 1721, 2201, 2252
	\@@_pgf_rect_node:nnn 259, 3687

\@@_pgf_rect_node:nnnnn	\@@_set_initial_coords:
..... 234, 3558, 3585, 3648, 3682 1962, 1981
\c_@@_pgfortikzpicture_tl	\@@_set_initial_coords_from_anchor:n .
..... 31, 35, 1787, 2793, 3312	... 1972, 2023, 2065, 2115, 2125, 2192, 2243
\@@_picture_position:	\@@_set_seq_of_str_from_clist:Nn 3869, 3874
..... 945, 953, 961	\@@_set_size:n
\g_@@_pos_of_blocks_seq 3733, 3742
..... 221, 923, 2566, 3048, 3070, 3127, 3281, 3610	\c_@@_siunitx_loaded_bool
\g_@@_pos_of_xdots_seq 135, 139, 144, 513, 887
..... 222, 1953, 3049, 3072, 3129, 3177	\l_@@_small_bool
\@@_pre_array: 474,
..... 745, 1032	516, 522, 536, 557, 757, 1492, 1534, 1743
\c_@@_preamble_first_col_tl	\@@_standard_cline
..... 1049, 1485 100, 798
\c_@@_preamble_last_col_tl	\@@_standard_cline:w
..... 1061, 1525 100, 101
\@@_pred:n	\l_@@_standard_cline_bool ...
..... 104, 126, 1634, 3080, 3093, 3137, 3150 283, 346, 797
\@@_put_box_in_flow:	\c_@@_standard_tl 293, 294, 2277, 3344, 3374
..... 1206, 1216, 1287	\l_@@_stop_loop_bool
\@@_put_box_in_flow_bis:nn 1815, 1816,
..... 1205, 1254	1848, 1861, 1870, 1883, 1884, 1916, 1929, 1938
\@@_put_box_in_flow_i:	\@@_succ:n
..... 1222, 1224 121,
\@@_qpoint:n	125, 689, 695, 1244, 1447, 1453, 1458,
..... 178, 1125, 1127, 1154, 1156, 1242, 1244,	1459, 1467, 1472, 1477, 1478, 1714, 2026,
1247, 2016, 2020, 2026, 2030, 2060, 2068,	2068, 2080, 2128, 2138, 2195, 2197, 2240,
2078, 2080, 2122, 2128, 2136, 2138, 2187,	2246, 2825, 2838, 2859, 2876, 2902, 2908,
2189, 2195, 2197, 2238, 2240, 2246, 2248,	2942, 2944, 3009, 3013, 3033, 3038, 3064,
2801, 2804, 2821, 2825, 2838, 2840, 2857,	3110, 3123, 3133, 3167, 3331, 3372, 3524,
2859, 2872, 2876, 2896, 2902, 2904, 2908,	3528, 3538, 3542, 3642, 3644, 3679, 3815, 3817
2931, 2933, 2942, 2944, 3007, 3009, 3015,	\l_@@_suffix_tl
3035, 3038, 3106, 3108, 3110, 3163, 3165, 3486, 3497,
3167, 3320, 3324, 3331, 3367, 3370, 3372,	3507, 3510, 3559, 3567, 3568, 3586, 3594, 3595
3464, 3474, 3638, 3640, 3642, 3644, 3664,	\c_@@_table_collect_begin_tl .
3679, 3700, 3701, 3707, 3810, 3812, 3815, 3817 152, 154, 167
\l_@@_radius_dim	\c_@@_table_print_tl
..... 290, 291, 876, 155, 156, 169
1745, 2034, 2035, 2450, 3298, 3322, 3368, 3369	\@@_test_if_cell_in_a_block:nn
\l_@@_real_left_delim_dim 1256, 1271, 1286 3222, 3240, 3258, 3278, 3284
\l_@@_real_right_delim_dim 1257, 1283, 1289	\@@_test_if_cell_in_block:nnnnnnn ...
\@@_rectanglecolor 3282, 3285
..... 974, 2919	\@@_test_if_hline_in_block:nnnn
\@@_renew_NC@rewrite@S: 3071, 3073, 3179
..... 160, 887	\@@_test_if_math_mode:
\l_@@_renew_dots_bool 186, 908, 1579, 1587, 1595, 1603, 1611
..... 379, 816, 3852	\@@_test_if_vline_in_block:nnnn
\@@_renew_matrix: 3128, 3130, 3193
..... 476, 3715, 3854	\l_@@_the_array_box
\@@_restore_iRow_jCol: 1046, 1073, 1135, 1139, 1165, 1194
..... 1780, 1803	\c_@@_tikz_loaded_bool 22, 30, 965, 1762, 3351
\c_@@_revtex_bool	\l_@@_tikz_tl
..... 39, 41, 44, 668 3620, 3647
\l_@@_right_delim_dim	\l_@@_type_of_col_tl
..... 1034, 1038, 1044, 1288, 1559 509, 510, 511, 512, 514, 1620, 1622
\l_@@_right_delim_tl	\c_@@_types_of_matrix_seq
..... 901, 3342 3874, 3881
\l_@@_right_margin_dim	\@@_update_for_first_and_last_row: ..
..... 311, 388, 1096, 1560, 2145, 3336, 3550 594, 621, 987, 1510, 1551
\@@_rotate:	\@@_vdottedline:n
..... 811, 2733 882, 3347
\@@_rotate_i:	\@@_vdottedline_i:n
..... 2737, 2739 3354, 3359, 3363
\@@_rotate_ii:	\@@_vline:
..... 2736, 2739, 2740 921, 2996
\@@_rotate_iii:	\l_@@_vlines_bool
..... 2740, 2741 302, 371, 375, 1055, 1067, 1079, 1098, 1754
\g_@@_row_of_col_done_bool	\g_@@_width_first_col_dim
..... 208, 710, 902, 1364 219, 1118, 1511, 1512
\g_@@_row_total_int	\g_@@_width_last_col_dim
..... 829, 1113, 218, 1210, 1552, 1553
1147, 1235, 1679, 1686, 1693, 1709, 2648,	\l_@@_x_final_dim
3418, 3432, 3459, 3554, 3630, 3653, 3668, 3976 212,
\@@_rowcolor	1969, 2027, 2028, 2069, 2070, 2118, 2140,
..... 975, 2814, 2959, 2960	2148, 2152, 2156, 2158, 2163, 2165, 2198,
\@@_rowcolors	2207, 2215, 2249, 2258, 2266, 2304, 2318,
..... 976, 2954	2327, 2363, 2415, 2431, 2805, 3332, 3343, 3369
\g_@@_rows_seq .	\l_@@_x_initial_dim
..... 1317, 1319, 1321, 1323, 1325 210, 1964, 2017, 2018, 2061, 2062,
\l_@@_rules_color_tl 2118, 2139, 2140, 2147, 2152, 2156, 2158,
..... 207, 339, 928, 929	
\@@_set_CT@arc@:	
..... 129, 929	
\@@_set_CT@arc@_i:	
..... 130, 131	
\@@_set_CT@arc@_ii:	
..... 130, 133	
\@@_set_final_coords:	
..... 1967, 1992	
\@@_set_final_coords_from_anchor:n .	
..... 1983, 2033, 2073, 2116, 2131, 2200, 2251	

<code>\cs_if_exist:NTF</code>	<code>\exp_args:Nnc</code> 3396
. . . 750, 753, 912, 915, 1000, 1007, 1018,	<code>\exp_args:NNV</code> 1319,
1025, 1805, 1806, 1851, 1864, 1919, 1932,	2460, 2475, 2490, 2505, 2527, 2586, 2662, 2760
2614, 2632, 2689, 2707, 3394, 3437, 3655, 3670	<code>\exp_args:Nnx</code> 1622
<code>\cs_if_exist_p:N</code> 323, 3225, 3243, 3261	<code>\exp_args:No</code> 2284
<code>\cs_if_free:Ntf</code>	<code>\exp_args:Nv</code> 869, 1295, 1322, 1324
. 725, 1996, 2040, 2095, 2173, 2224	<code>\exp_args:Nx</code> 3647
<code>\cs_if_free_p:N</code> 2780, 2782	<code>\exp_not:N</code> 31, 32, 35, 36, 3310, 3311
<code>\cs_new_protected:Npx</code> 1785, 2791, 3308	<code>\exp_not:n</code> 663, 2596, 2672, 3615
<code>\cs_set:Npn</code> 82, 83, 88, 100,	<code>\expandafter</code> 726
101, 111, 113, 114, 132, 134, 673, 727, 759,	<code>\ExplSyntaxOff</code> 1675, 1696, 1719, 3411
762, 1810, 1872, 1940, 2656, 2731, 3579, 3580	<code>\ExplSyntaxOn</code> 1661, 1682, 1701, 3404
<code>\cs_set_protected:Npn</code> 3767	
	F
D	<code>\fi</code> 90
<code>\Ddots</code> 803, 2509, 2510, 2515, 2516	fi commands:
<code>\ddots</code> 793, 821	<code>\fi:</code> 190
<code>\diagbox</code> 815	<code>\firsthline</code> 795
dim commands:	<code>\fontdimen</code> 1249
<code>\dim_add:Nn</code> 3548	fp commands:
<code>\dim_compare:nNnTF</code> 85,	<code>\fp_eval:n</code> 2322
1404, 1557, 2158, 3461, 3471, 3662, 3677, 3796	<code>\fp_to_dim:n</code> 2359
<code>\dim_max:nn</code> 3450, 3454	<code>\futurelet</code> 95
<code>\dim_min:nn</code> 3442, 3446	
<code>\dim_ratio:nn</code> 2216, 2267,	G
2383, 2388, 2399, 2407, 2416, 2421, 2432, 2440	group commands:
<code>\dim_set:Nn</code> 3423, 3430, 3441, 3445,	<code>\group_insert_after:N</code>
3449, 3453, 3465, 3466, 3475, 3476, 3520, 3533	2736, 2737, 2739, 2740, 3790, 3791, 3793, 3794
<code>\dim_set_eq:NN</code> 3421, 3428, 3528, 3542	
<code>\dim_sub:Nn</code> 3545	H
<code>\dim_use:N</code>	<code>\halign</code> 788
. 3462, 3472, 3523, 3524, 3536, 3537,	<code>\hbox</code> 682, 1189, 1369, 1387, 1414, 1418, 1444
3560, 3561, 3562, 3563, 3587, 3588, 3589, 3590	hbox commands:
<code>\dim_zero_new:N</code> 3420, 3422, 3427, 3429	<code>\hbox:n</code> 51, 53, 56
<code>\c_max_dim</code> 3421, 3423,	<code>\hbox_overlap_left:n</code> 1513
3428, 3430, 3462, 3472, 3652, 3662, 3667, 3677	<code>\hbox_overlap_right:n</code> 1555
<code>\l_tmpc_dim</code> 214, 2823, 2824, 2843,	<code>\hbox_set:Nn</code> 1041, 1043, 1182, 1258, 1273, 3634
2874, 2875, 2879, 2906, 2907, 2911, 2935,	<code>\hbox_set:Nw</code> . 553, 841, 855, 1073, 1490, 1532
2936, 2947, 3016, 3017, 3018, 3111, 3113,	<code>\hbox_set_end:</code> 614, 847, 861, 1099, 1509, 1550
3168, 3170, 3643, 3650, 3684, 3816, 3819, 3827	<code>\hbox_to_wd:nn</code> 252, 277
<code>\l_tmpd_dim</code> 215, 2841, 2843, 2877,	<code>\Hdotsfor</code> 807, 3895
2880, 2909, 2912, 2945, 2948, 3645, 3650,	<code>\hdotsfor</code> 824
3667, 3674, 3677, 3680, 3684, 3818, 3819, 3831	<code>\hdottedline</code> 805
<code>\dotfill</code> 813, 3785	<code>\hfil</code> 862
<code>\dots</code> 823	<code>\hfill</code> 106, 119
<code>\draw</code> 2289	<code>\hline</code> 88, 795, 796
	<code>\hrule</code> 92, 106, 119, 718
E	<code>\hskip</code> 91
<code>\egroup</code> 1214	<code>\Hspace</code> 806
else commands:	<code>\hspace</code> 2553
<code>\else:</code> 188	<code>\hss</code> 862
<code>\endarray</code> 1299, 1327	
<code>\endBNiceMatrix</code> 3731	I
<code>\endbNiceMatrix</code> 3728	<code>\ialign</code> 679, 762, 787, 1758
<code>\endNiceArray</code> 1652	<code>\Iddots</code> 804, 2531, 2532, 2537, 2538
<code>\endNiceArrayWithDelims</code>	<code>\iddots</code> 46, 794, 822
. 1574, 1582, 1590, 1598, 1606, 1614	if commands:
<code>\endpgfscope</code> 2351, 3830	<code>\if_mode_math:</code> 188
<code>\endpNiceMatrix</code> 3719	<code>\ifnum</code> 90
<code>\endVNiceMatrix</code> 3725	<code>\ifstandalone</code> 915
<code>\endvNiceMatrix</code> 3722	int commands:
<code>\everycr</code> 107, 122, 774	<code>\int_case:nnTF</code> 2507, 2513, 2529, 2535
exp commands:	<code>\int_compare:nNnTF</code> 103, 104, 116,
<code>\exp_after:wN</code> 164, 929	550, 551, 561, 568, 604, 715, 717, 873, 875,

878, 940, 942, 985, 995, 1014, 1100, 1104, 1114, 1115, 1130, 1159, 1320, 1348, 1349, 1535, 1826, 1833, 1837, 1839, 1894, 1901, 1905, 1907, 2003, 2047, 2102, 2141, 2143, 2561, 2648, 2723, 2744, 2836, 2870, 2938, 2940, 2985, 2986, 2993, 2994, 3181, 3183, 3185, 3187, 3195, 3197, 3199, 3201, 3697, 3745, 3747, 3749, 3753, 3755, 3757, 3759, 3761	\NewDocumentCommand 499, 2460, 2475, 2490, 2505, 2527, 2586, 2662, 2760, 2814, 2850, 2886, 2919, 2954, 2964, 3738, 3778
\int_compare_p:n 2898, 2899, 2928, 2929, 3289, 3290, 3291, 3292	\NewDocumentEnvironment 897, 1291, 1301, 1567, 1575, 1583, 1591, 1599, 1607, 1617, 1645, 3387
\int_gadd:Nn 2574	\NewExpandableDocumentCommand 176, 3598
\int_gincr:N 549, 577, 918, 1440, 1530, 2203, 2254, 3389	\NiceArray 1650
\int_if_even:nTF 2970	\NiceArrayWithDelims 1572, 1580, 1588, 1596, 1604, 1612
\int_step_inline:nn 2966, 2968	\NiceMatrixLastEnv 176
\int_step_inline:nnnn 3220, 3238, 3253, 3256	\NiceMatrixOptions 499, 4034
iow commands:	\NiceMatrixoptions 3996
\iow_now:Nn 63, 1701, 1702, 1704, 1719	\noalign 85, 90, 109, 706, 770, 3298
\iow_shipout:Nn 1661, 1662, 1669, 1675, 1682, 1683, 1690, 1696, 3404, 3405, 3411	\normalbaselines 756
	\nulldelimiterspace 1272, 1284
	\numexpr 125, 126
	O
	\omit 103, 1351, 1363, 1439
	\OnlyMainNiceMatrix 812, 2977
K	P
\kern 56	peek commands:
keys commands:	\peek_meaning:NTF 130, 728
\keys_define:nn 318, 337, 344, 398, 433, 469, 501, 520, 529, 540, 3378, 3618, 3850	\peek_meaning_ignore_spaces:NTF 1293
\l_keys_key_tl 3953, 4028, 4033, 4074, 4120, 4169	\peek_meaning_remove_ignore_spaces:NTF 120
\keys_set:nn 352, 500, 925, 926, 1621, 1648, 2006, 2050, 2105, 2177, 2228, 2651, 2726, 2763, 3391, 3628	\peek_remove_spaces:n 2559
\l_keys_value_tl 3992, 3999, 4214	\pgfextracty 3700
	\pgfgetlastxy 270
	\pgfpathcircle 2448
	\pgfpathlineto 3018, 3040, 3113, 3170, 3819
	\pgfpathmoveto 3017, 3037, 3112, 3169, 3814
	\pgfpathrectanglecorners 2842, 2878, 2910, 2946
L	\pgfpointhead 268
\lasthline 796	\pgfpointanchor 179, 1974, 1985, 3440, 3448, 3658, 3673, 3689, 3690
\Ldots 800, 2463, 2466	\pgfpointdiff 269, 953, 961
\ldots 790, 818	\pgfpointlineatime 2316
\leaders 106, 119	\pgfpointorigin 1357, 1473
\left 1185, 1261, 1276	\pgfpointscale 268
legacy commands:	\pgfpointshapeborder 2801, 2804
\legacy_if:nTF 421	\pgfrememberpicturerepositiononpagetrue 582, 635, 694, 1356, 1377, 1395, 1426, 1452, 1471, 1794, 2275, 2353, 2800, 3003, 3028, 3102, 3159, 3319, 3366, 3483, 3493, 3504, 3636, 3809
\line 1774, 3983	\pgfscope 2313, 3826
	\pgfset 237, 262, 636, 3647, 3825
	\pgfsetbaseline 634
M	\pgfsetlinewidth 3005, 3030, 3104, 3161
\makebox 848, 863	\pgfsetrectcap 3006, 3105, 3162
\mathinner 48	\pgfsetroundcap 3822
mode commands:	\pgftransformrotate 2320
\mode_leave_vertical: 739, 907	\pgftransformshift 243, 268, 2314, 3702, 3707, 3827, 3831
msg commands:	\pgfusepath 2340, 2350
\msg_error:nn 12	\pgfusepathqfill .. 2454, 2846, 2882, 2915, 2949
\msg_error:nnn 13	\pgfusepathqstroke 3020, 3042, 3114, 3171, 3823
\msg_error:nnnn 14, 3632	\phantom 2472, 2487, 2502, 2524, 2546
\msg_fatal:nn 15	\pNiceMatrix 3718
\msg_fatal:nnn 16	
\msg_new:nnn 17	prg commands:
\msg_new:nnnn 18	\prg_do_nothing: 148, 157, 770, 1775
\msg_redirect_name:nnn 20	
\multicolumn 809, 2555, 2579, 2599	
\multispan 104, 105, 117, 118	
\myfiledate 6	
\myfileversion 7	
N	
\newcolumnntype 869	

\prg_replicate:nn
 ... 1435, 1436, 2599, 3748, 3751, 3754, 3760
 \ProcessKeysOptions 3859
 \ProvideDocumentCommand 46
 \ProvidesExplPackage 4

Q

quark commands:

\q_stop 100, 101, 113,
 114, 131, 133, 929, 1311, 1314, 2754, 2768,
 2769, 2809, 2831, 2832, 2865, 2866, 2895,
 2926, 2937, 3577, 3584, 3599, 3600, 3733, 3742

R

\rectanglecolor 974
 \relax 125, 126
 \renewcommand 162
 \RenewDocumentEnvironment
 3717, 3720, 3723, 3726, 3729
 \RequirePackage 1, 3, 9, 10, 11
 \right 1201, 1268, 1280
 \rotate 811
 \rowcolor 975
 \rowcolors 976

S

\scriptstyle
 557, 1492, 1534, 2297, 2301, 2336, 2346

seq commands:

\seq_clear:N 922, 923, 3862
 \seq_clear_new:N 1703, 3209
 \seq_count:N 1321
 \seq_gclear:N 3177
 \seq_gclear_new:N 826, 827, 1317, 1333
 \seq_gpop_left:NN 1323, 1335
 \seq_gput_left:Nn 426, 2563, 2565, 3610, 3611
 \seq_gput_right:Nn 1953, 2566
 \seq_gset_from_clist:Nn 1706
 \seq_gset_split:Nnn 1319, 1334
 \seq_if_empty:NTF 1759
 \seq_if_empty_p:N 3048, 3049
 \seq_if_exist:NTF 932
 \seq_if_in:NnTF 424, 3078,
 3084, 3091, 3135, 3141, 3148, 3443, 3451, 3881
 \seq_item:Nn 936, 939, 948, 949, 956, 957
 \seq_map_function:NN 1325
 \seq_map_inline:Nn
 1337, 3070, 3072, 3127, 3129, 3281, 3624, 3863
 \seq_mapthread_function:NNN 3572
 \seq_new:N 182, 220, 221, 222
 \seq_put_left:Nn 3865
 \seq_put_right:Nn 3270
 \seq_set_eq:NN 3867
 \seq_set_from_clist:Nn 3871
 \seq_use:Nnnn 4223
 \l_tmpa_seq 3862, 3865, 3867

skip commands:

\skip_gadd:Nn 1411
 \skip_gset:Nn 1402
 \skip_gset_eq:NN 1409, 1410
 \skip_horizontal:N 876, 1082, 1084, 1085,
 1096, 1097, 1098, 1117, 1118, 1192, 1193,
 1196, 1197, 1210, 1211, 1285, 1286, 1288,
 1289, 1352, 1371, 1373, 1389, 1391, 1413,

1420, 1422, 1441, 1446, 1448, 1469, 1481,
 1518, 1519, 1520, 1522, 1554, 1559, 1560, 1561

\skip_vertical:N
 109, 688, 690, 1188, 1199, 3298
 \skip_vertical:n 2749
 \g_tmpa_skip 1402, 1409, 1410, 1411, 1413, 1441
 \c_zero_skip 775

\space 200, 201

str commands:

\c_backslash_str 200
 \c_colon_str 498
 \str_case:nnTF 1226
 \str_gclear:N 1779
 \str_gset:Nn 904, 1571,
 1578, 1586, 1594, 1602, 1610, 1619, 1647, 3769
 \str_if_empty:NTF
 586, 647, 697, 903, 998, 1016, 1358,
 1380, 1398, 1429, 1455, 1474, 1570, 1577,
 1585, 1593, 1601, 1609, 1667, 1688, 3564, 3591
 \str_if_eq:nnTF 71, 199, 416,
 482, 677, 1122, 1138, 1141, 1220, 1341, 3843
 \str_if_eq_p:nn 324
 \str_lowercase:n 848, 863
 \str_new:N 194, 195, 296, 307, 497
 \str_set:Nn
 196, 297, 410, 411, 412, 423, 491, 1142
 \str_set_eq:NN 427, 498
 \l_tmpa_str 423, 424, 426, 427

T

\tabcolsep 1077,
 1192, 1196, 2019, 2029, 2063, 2071, 3328, 3335

\tabskip 775

TeX and L^AT_EX commands:

\@BTnormal 748
 \@acol 672
 \@acoll 670
 \@acolr 671
 \@addtopreamble 921
 \@array@array 674
 \@arrayacol 670, 671, 672
 \@arrayrule 921
 \@arstrutbox 579,
 580, 739, 742, 777, 779, 781, 784, 786, 2749
 \@currenvir 3843
 \@halignto 673
 \@height 93, 106, 119
 \@ifclassloaded 40, 43
 \@ifnextchar 831
 \@ifpackageloaded 25, 28, 62, 78, 138
 \@mainaux 63,
 1661, 1662, 1669, 1675, 1682, 1683, 1690,
 1696, 1701, 1702, 1704, 1719, 3404, 3405, 3411
 \@temptokena 147, 150, 164, 166
 \@width 93
 \@xhline 96
 \bBigg@ 1041, 1043
 \c@MaxMatrixCols 1633, 3909
 \col@sep .. 1352, 1411, 1469, 1481, 1522, 1554
 \CT@arc 82, 83
 \CT@arc@ ... 81, 86, 94, 106, 119, 132, 134,
 911, 1781, 2997, 3001, 3027, 3061, 3365, 3821
 \CT@everycr 768
 \CT@row@color 770

5	The rules	5
5.1	The thickness and the color of the rules	5
5.2	A remark about <code>\cline</code>	5
5.3	The keys <code>hlines</code> and <code>vlines</code>	5
5.4	The key <code>hvlines</code>	6
5.5	The key <code>hvlines-except-corners</code>	6
5.6	The command <code>\diagbox</code>	7
5.7	Dotted rules	7
6	The color of the rows and columns	8
7	The width of the columns	10
8	The exterior rows and columns	11
9	The continuous dotted lines	13
9.1	The option <code>nullify-dots</code>	14
9.2	The command <code>\Hdotsfor</code>	14
9.3	How to generate the continuous dotted lines transparently	15
9.4	The labels of the dotted lines	15
9.5	Customization of the dotted lines	16
9.6	The dotted lines and the key <code>hvlines</code>	17
10	The code-after	17
11	Other features	17
11.1	Use of the column type <code>S</code> of <code>siunitx</code>	17
11.2	Alignment option in <code>{NiceMatrix}</code>	18
11.3	The command <code>\rotate</code>	18
11.4	The option <code>small</code>	18
11.5	The counters <code>iRow</code> and <code>jCol</code>	19
11.6	The option <code>light-syntax</code>	20
11.7	The environment <code>{NiceArrayWithDelims}</code>	20
12	Utilisation of Tikz with <code>nicematrix</code>	20
12.1	The nodes corresponding to the contents of the cells	20
12.2	The “medium nodes” and the “large nodes”	21
12.3	The “row-nodes” and the “col-nodes”	22
13	Technical remarks	23
13.1	Definition of new column types	23
13.2	Diagonal lines	24
13.3	The “empty” cells	24
13.4	The option <code>exterior-arraycolsep</code>	25
13.5	Incompatibilities	25
14	Examples	25
14.1	Dotted lines	25
14.2	Dotted lines which are no longer dotted	27
14.3	Width of the columns	27
14.4	How to highlight cells of the matrix	28
14.5	Direct use of the Tikz nodes	31
15	Implementation	32
16	History	130
	Index	135