

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

August 19, 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.

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

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

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

This package can be used with `xelatex`, `lualatex`, `pdflatex` but also by the classical workflow `latex-dvips-ps2pdf` (or Adobe Distiller).

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).

Important

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

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

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

*This document corresponds to the version 5.2 of `nicematrix`, at the date of 2020/08/19.

1 The environments of this package

The package `nicematrix` defines the following new environments.

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

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

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

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

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

Important

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

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

2 The vertical space between the rows

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

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

Inspired by the package `cellspace` which deals with that problem, the package `nicematrix` provides two keys `cell-space-top-limit` and `cell-space-bottom-limit` similar to the parameters `\cellspacetoplimit` and `\cellspacebottomlimit` of `cellspace`. The initial value of these parameters is 0 pt in order to have for the environments of `nicematrix` the same behaviour as those of `array` and `amsmath`. However, a value of 1 pt would probably be a good choice and we suggest to set them with `\NiceMatrixOptions`.¹

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

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

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

3 The vertical position of the arrays

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

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

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

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

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

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

1. an item

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

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

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

1. an item

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

New 5.2 It's also possible to use the key `baseline` to align a matrix on an horizontal rule (drawn by `\hline`). In this aim, one should give the value `line-i` where i is the number of the row following the horizontal rule.

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

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

$$A = \begin{pmatrix} \frac{1}{A} & \frac{1}{B} & 0 & 0 \\ \frac{1}{C} & \frac{1}{D} & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & D & D \end{pmatrix}$$

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.

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

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

It's also possible to use the command `\Block` in mathematical matrices.

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

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

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`). However, there is some small differences with the classical environments.

5.1 Some differences with the classical environments

5.1.1 The vertical rules

In the environments of `nicematrix`, the vertical rules specified by `|` in the preambles of the environments are never broken, even by an incomplete row or by a double horizontal rule specified by `\hline\hline` (there is no need to use `hhline`).²

```
\begin{NiceTabular}{|c|c|} \hline
First & Second \\ \hline\hline
Peter \\ \hline
Mary & George \\ \hline
\end{NiceTabular}
```

First	Second
Peter	
Mary	George

New 5.2 However, the vertical rules are not drawn in the blocks.

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

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

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

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

```
\newcolumntype{I}{!{\vrule}}
```

However, in this case, it is probably more clever to add a command `\OnlyMainNiceMatrix` (cf. p. 31):

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

²This is the behaviour since the version 5.1 of `nicematrix`. Prior to that version, the behaviour was the standard behaviour of `array`.

5.1.2 The command `\cline`

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

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

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

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

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

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

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

5.2 The thickness and the color of the rules

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

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

With `nicematrix`, it’s possible to specify the color of the rules even when `colortbl` is not loaded. For sake of compatibility, the command is also named `\arrayrulecolor`. The environments of `nicematrix` also provide a key `rules/color` to fix the color of the rules in the current environment.

```
\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 31.

5.3 The keys `hlines` and `vlines`

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

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

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

5.4 The key hvlines

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

```
\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 the virtual blocks determined by dotted lines) and excepted in the empty corners.

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

				A	
		A	A	A	
			A		
		A	A	A	A
A	A	A	A	A	A
A	A	A	A	A	A
		B		A	
				A	
	A	A	A		

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

New 5.2 It’s possible to give as value to the key `hvlines-except-corners` a list of the corners to take into consideration. The corners are designed by NW, SW, NE and SE (*north west*, *south west*, *north east* and *south east*).

```

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

```

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

5.6 The command `\diagbox`

The command `\diagbox` (inspired by the package `diagbox`), allows, when it is used in a cell, to slash that cell diagonally downwards.³

```

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

```

$\begin{smallmatrix} y \\ x \end{smallmatrix}$	<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>

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

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)

```

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

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 rule⁴. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

³The author of this document considers that type of construction as graphically poor.

⁴In fact, this is true only for `\hline` and “|” but not for `\cline`: cf p. 6

6 The color of the rows and columns

6.1 Use of colortbl

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

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

However, there is two drawbacks:

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

As for this phenomenon, Adobe Reader gives better results than MuPDF.

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

6.2 The tools of nicematrix in the code-before

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

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

All these commands accept an optional argument (between square brackets and in first position) which is the color model for the specification of the colors.

- The command `\cellcolor` takes its name from the command `\cellcolor` of `colortbl`.

This command takes in as mandatory arguments a color and a list of cells, each of which with the format *i-j* where *i* is the number of row and *j* the number of column of the cell.

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

a	b	c
e	f	g
h	i	j

A command `\cellcolor` generates only one instruction `fill` (coded `f`) in the resulting PDF.

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

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

a	b	c
e	f	g
h	i	j

- The command `\rowcolor` takes its name from the command `\rowcolor` of `colortbl`. Its first mandatory argument is the color and the second is a comma-separated list of rows or interval of rows with the form *a-b* (an interval of the form *a-* represent all the rows from the row *a* until the end).

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

<i>a</i> ₁	<i>b</i> ₁	<i>c</i> ₁
<i>a</i> ₂	<i>b</i> ₂	<i>c</i> ₂
<i>a</i> ₃	<i>b</i> ₃	<i>c</i> ₃
<i>a</i> ₄	<i>b</i> ₄	<i>c</i> ₄
<i>a</i> ₅	<i>b</i> ₅	<i>c</i> ₅
<i>a</i> ₆	<i>b</i> ₆	<i>c</i> ₆
<i>a</i> ₇	<i>b</i> ₇	<i>c</i> ₇
<i>a</i> ₈	<i>b</i> ₈	<i>c</i> ₈
<i>a</i> ₉	<i>b</i> ₉	<i>c</i> ₉
<i>a</i> ₁₀	<i>b</i> ₁₀	<i>c</i> ₁₀

A command `\rowcolor` generates only one instruction `fill` (coded `f`) in the resulting PDF.

- 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`⁵. 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

New 5.2 There is a key `respect-blocks` for the instruction `\rowcolor`. With that key, the “rows” alternately colored may extend over several row if they have to incorporate blocks.

⁵The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`.

```

\begin{NiceTabular}{cr}[hvlines,code-before =
\rowcolors{1}{blue!10}{}[respect-blocks]]
\Block{2-1}{John} & 12 \\
& 13 \\
Stephen & 8 \\
\Block{3-1}{Sarah} & 18 \\
& 17 \\
& 15 \\
Ashley & 20 \\
Henry & 14 \\
\Block{2-1}{Madison} & 15 \\
& 19
\end{NiceTabular}

```

John	12
	13
Stephen	8
Sarah	18
	17
	15
Ashley	20
Henry	14
Madison	15
	19

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

```

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

```

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

We have used the key `r` which aligns all the columns rightwards (cf. p. 25).

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

```

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

```

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

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

6.3 Color tools with the syntax of `colortbl`

It's possible to access the preceding tools with a syntax close to the syntax of `colortbl`. For that, one must use the key `colortbl-like` in the current environment.⁶ There are three commands available (they are inspired by `colortbl` but are *independent* of `colortbl`):

- `\cellcolor` which colorizes a cell;
- `\rowcolor` which must be used in a cell and which colorizes the end of the row;

⁶As of now, this key is not available in `\NiceMatrixOptions`.

- `\columncolor` which must be used in the preamble of the environment with the same syntax as the corresponding command of `colortbl`⁷.

```
\begin{NiceTabular}[colortbl-like]{*{2}->{\columncolor{blue!15}}c}c}
\toprule
\rowcolor{red!15}
Last name & First name & Birth day \\
\midrule
Achard & Jacques & 5 juin 1962 \\
Lefebvre & Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}
```

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

Each instruction `\cellcolor`, `\rowcolor` or `\columncolor` will generate an instruction `fill` (coded `f`) in the resulting PDF. In cases of juxtaposed colored rectangles, one may have a thin white color line in some PDF viewers⁸ (between the two first columns in the above example). In you want to avoid this problem, you should use the tools in the `code-before`. That's what we do with the following code.

```
\begin{NiceTabular}[colortbl-like]{ccc}%
[code-before = \columncolor{blue!15}{1,2}]
\toprule
\rowcolor{red!15}
Last name & First name & Birth day \\
\midrule
Achard & Jacques & 5 juin 1962 \\
Lefebvre & Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}
```

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

⁷Unlike the command `\columncolor` of `colortbl`, this command `\columncolor` can appear within another command, itself used in the preamble.

⁸For example SumatraPDF v3.1.2, which uses MuPDF of Artifex Software, whose PDF renderer is called Fitz.

7 The width of the columns

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

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

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

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

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

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

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

It's possible to give the special value `auto` to the option `columns-width`: all the columns of the array will have a width equal to the widest cell of the array.⁹

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

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

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. 4).

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

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

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

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

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

Several compilations may be necessary to achieve the job.

8 The exterior rows and columns

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

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

```

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

```

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

The dotted lines have been drawn with the tools presented p. 15.

We have several remarks to do.

- For the environments with an explicit preamble (i.e. `{NiceArray}` and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type `r` for the first column and `l` for the last one.
- One may wonder how `nicematrix` determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
 - For the environments with explicit preamble, like `{NiceTabular}` and `{pNiceArray}`, the number of columns can obviously be computed from the preamble.
 - When the option `light-syntax` (cf. p. 27) is used, `nicematrix` has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (but not the number of columns).
 - In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

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

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

```

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

```

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

Remarks

- As shown in the previous example, the horizontal and vertical rules doesn't extend in the exterior rows and columns.
However, if one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 31.
- 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. 13) 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

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. 32).

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}
```

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

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

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

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

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

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

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

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

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

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\Vdots` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.¹⁴

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

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

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

¹³It's also possible to change the color of all theses dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 19.

¹⁴In `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. 13.

9.1 The option nullify-dots

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used when the package `colortbl` is loaded (but you might have problem if you use `\rowcolor` on the same row as `\Hdotsfor`).

The package `nicematrix` also provides a command `\Vdotsfor` similar to `\Hdotsfor` but for the vertical dotted lines. The following example uses both `\Hdotsfor` and `\Vdotsfor`:

```
\begin{bNiceMatrix}
C[a_1,a_1] & \Cdots & C[a_1,a_n]
& \hspace*{20mm} & C[a_1,a_1^{(p)}] & \Cdots & C[a_1,a_n^{(p)}] \\
\Vdots & \Ddots & \Vdots
& \Hdotsfor{1} & \Vdots & \Ddots & \Vdots \\
C[a_n,a_1] & \Cdots & C[a_n,a_n]
& & C[a_n,a_1^{(p)}] & \Cdots & C[a_n,a_n^{(p)}] \\
\rule{0pt}{15mm} & \Vdotsfor{1} & & \Ddots & & \Vdotsfor{1} \\
C[a_1^{(p)},a_1] & \Cdots & C[a_1^{(p)},a_n]
& & C[a_1^{(p)},a_1^{(p)}] & \Cdots & C[a_1^{(p)},a_n^{(p)}] \\
& & C[a_1^{(p)},a_1^{(p)}] & \Cdots & C[a_1^{(p)},a_n^{(p)}] \\
\Vdots & \Ddots & \Vdots
& \Hdotsfor{1} & \Vdots & \Ddots & \Vdots \\
C[a_n^{(p)},a_1] & \Cdots & C[a_n^{(p)},a_n]
& & C[a_n^{(p)},a_1^{(p)}] & \Cdots & C[a_n^{(p)},a_n^{(p)}] \\
\end{bNiceMatrix}
```

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

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`.¹⁵

- 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}
```

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

¹⁵The 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.)

9.4 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. 20) 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`.

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

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

9.5 Customization of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and by the command `\line` in the `code-after` which is described p. 20) 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. 14.

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).¹⁶

```
\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 <`”).

¹⁶The 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.

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

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

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

9.6 The dotted lines and the rules

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

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

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

10 The code-after

The option `code-after` may be used to give some code that will be executed after the construction of the matrix.¹⁷

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

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

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

For the legibility of the code, an alternative syntax is provided: it's possible to give the instructions of the `\code-after` at the end of the environment, after the keyword `\CodeAfter`¹⁸. For an example, cf. p. 37.

¹⁷There is also a key `code-before` described p. 9.

¹⁸In some circumstances, one must put `\omit \CodeAfter`. `\omit` is a keyword of TeX which cancels the pattern of the current cell.

11 The notes in the tabulars

11.1 The footnotes

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

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

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

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

11.2 The notes of tabular

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

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

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

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

^a Achard is an old family of the Poitou.

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

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

Table 1: Use of `\tabularnote`^a

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

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

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

For an illustration of some of those remarks, see table 1, p. 22. This table has been composed with the following code.

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

11.3 Customisation of the tabular notes

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

- `notes/para`

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

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

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

We detail these keys.

- The key `notes/para` requires the composition of the notes (at the end of the tabular) in a single paragraph.
Initial value: `false`
That key is also available within a given environment.
- The key `notes/bottomrule` adds a `\bottomrule` of `booktabs` *after* the notes. Of course, that rule is drawn only if there is really notes in the tabular. The package `booktabs` must have been loaded (before or after the package `nicematrix`). If it is not, an error is raised.
Initial value: `false`
That key is also available within a given environment.
- The key `notes/style` is a command whose argument is specified by `#1` and which gives the style of numerotation of the notes. That style will be used by `\ref` when referencing a tabular note marked with a command `\label`. The labels formatted by that style are used, separated by commas, when the user puts several consecutive commands `\tabularnote`. The marker `#1` is meant to be the name of a LaTeX counter.
Initial value: `\textit{\alph{#1}}`
Another possible value should be a mere `\arabic{#1}`
- The key `notes/label-in-tabular` is a command whose argument is specified by `#1` which is used when formatting the label of a note in the tabular. Internally, this number of note has already been formatted by `notes/style` before sent to that command.
Initial value: `#1`
In French, it's a tradition of putting a small space before the label of note. That tuning could be achieved by the following code:

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

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

Initial value: `#1`

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

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

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

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

Initial value: `noitemsep , leftmargin = * , align = left , labelsep = 0pt`

This initial value contains the specification `align = left` which requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 22).

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

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

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

Initial value: *empty*

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

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

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

For an example of customization of the tabular notes, see p. 33.

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

If you wish to use the environment `{NiceTabular}` in an environment `{threeparttable}` of the eponymous package, you have to patch the environment `{threeparttable}` with the following code:

```
\makeatletter
\AtBeginEnvironment{threeparttable}{\TPT@hookin{NiceTabular}}
\makeatother
```

The command `\AtBeginEnvironment` is a command of the package `etoolbox` which must have been loaded previously.

12 Other features

12.1 Use of the column type S of siunitx

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

```


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


```

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

12.2 Alignment option in `{NiceMatrix}`

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

```


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


```

12.3 The command `\rotate`

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

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

```

\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}

```

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

12.4 The option `small`

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

```

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

```

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

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

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

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

12.5 The counters `iRow` and `jCol`

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column¹⁹. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `code-before` (cf. p. 9) and in the `code-after` (cf. p. 20), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

```

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

```

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

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

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

¹⁹We recall that the exterior “first row” (if it exists) has the number 0 and that the exterior “first column” (if it exists) has also the number 0.

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

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

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

12.6 The option `light-syntax`

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

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

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.²⁰

12.7 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.

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

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

13 Use of Tikz with `nicematrix`

13.1 The nodes corresponding to the contents of the cells

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

²⁰The 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.

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.

```

 $\begin{pNiceMatrix}[name=mymatrix]$ 
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. 37).

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

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

In fact, the package `nicematrix` can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option `create-medium-nodes` and the second ones with the option `create-large-nodes`.²¹

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.

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

²¹There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.²²

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

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.²³

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

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

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

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

Here is an array composed with the following code:

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

²²There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 14).

²³The options `left-margin` and `right-margin` take dimensions as values but, if no value is given, the default value is used, which is `\arraycolsep` (by default: 5 pt). There is also an option `margin` to fix both `left-margin` and `right-margin` to the same value.

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

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

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

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

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

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

14 API for the developpers

The package `nicematrix` provides two variables which are internal but public²⁴:

- **New 5.2** `\g_nicematrix_code_before_tl` ;
- `\g_nicematrix_code_after_tl`.

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

²⁴According to the LaTeX3 conventions, each variable with name beginning with `\g_nicematrix` ou `\l_nicematrix` is public and each variable with name beginning with `\g__nicematrix` or `\l__nicematrix` is private.

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

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

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

\NewDocumentCommand \hatchcell { ! 0 { black } }
{
  \tl_gput_right:Nx \g_nicematrix_code_before_tl
  {
    \__pantigny_hatchcell:nnn
    { \int_use:c { c@iRow } }
    { \int_use:c { c@jCol } }
    { #1 }
  }
}
\ExplSyntaxOff
```

Here is an example of use:

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

Tokyo	Paris	London
Lima	Oslo	Miami
Los Angeles	Madrid	Roma

15 Technical remarks

15.1 Definition of new column types

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

For example, one may wish to define a new column type `?` in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job²⁵:

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

The heavy vertical rule won't extend in the exterior rows.²⁶

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

²⁶Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

```

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

```

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

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

15.2 Diagonal lines

By default, all the diagonal lines²⁷ of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That’s why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

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

Example with parallelization (default):

```

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

```

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

```

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

```

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

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

The same example without parallelization:

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

15.3 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}

```

²⁷We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

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`.

15.4 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea²⁸. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`²⁹. The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

15.5 Incompatibilities

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

16 Examples

16.1 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 11 p. 21.

Let's consider that we wish to number the notes of a tabular with stars.³⁰

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument³¹

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

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

²⁸In 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).*

²⁹And 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

³⁰Of course, it's realistic only when there is very few notes in the tabular.

³¹In fact: the value of its argument.

by `\tabularnote` and, therefore, at the end of the tabular, its value is equal to the total number of tabular notes). We use the key `widest*` of `enumitem` in order to require a width equal to that value: `widest*=\value{tabularnote}`.

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

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

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

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

16.2 Dotted lines

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

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

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

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

```

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

```

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

An example with `\multicolumn`:

```

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

```

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

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 \\
 $\end{pNiceMatrix}$ 

```

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

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}

```


16.4 Width of the columns

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

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

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

16.5 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.³²

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}}

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

```

$$\begin{bmatrix} 0 \cdots \cdots 0 \\ \textcolor{red}{1} \cdots \cdots \textcolor{red}{1} \\ 0 \cdots \cdots 0 \end{bmatrix}$$

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

```

³²For the command `\cline`, see the remark p. 6.

```

{\cs_set:Npn\pgfsys@blend@mode#1{\special{ps:~/\tl_upper_case:n #1~.setblendmode}}}
\makeatother
\ExplSyntaxOff

```

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

```

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

```

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

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

```

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

```

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

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

```

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

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

```

We obtain the following matrix.

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

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

```

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

```

We obtain the following matrix.

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

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

```

\begin{pNiceArray}{>{\strut}cccc}[create-large-nodes,margin,extra-margin=2pt]
A_{11} & A_{12} & A_{13} & A_{14} \\
A_{21} & A_{22} & A_{23} & A_{24} \\
A_{31} & A_{32} & A_{33} & A_{34} \\
A_{41} & A_{42} & A_{43} & A_{44} \\
\CodeAfter
\tikz \path [name suffix = -large,fill = red!15, blend mode = multiply]
(1-1.north west)
|- (2-2.north west)
|- (3-3.north west)
|- (4-4.north west)
|- (4-4.south east)
|- (1-1.north west) ;
\end{pNiceArray}

```

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

16.6 Direct use of the Tikz nodes

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

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

```

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

\NiceMatrixOptions{nullify-dots}

```

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

```

$\begin{array}{cc}
& &

```

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


```

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

```

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

```

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

```

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

```

\begin{bNiceArray}{cc>{\strut}ccc}
    & & & & \\
    & & \Vdots & & \\
\Cdots & & c_{ij} & & \\
\\
\\
\end{bNiceArray}
\end{array}$

```

```

\end{NiceMatrixBlock}

```

```

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

```

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

17 Implementation

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

However, when the option `renew-dots` is used, the commands `\cdots`, `\ldots`, `\dots`, `\vdots`, `\ddots` and `\iddots` are redefined in the environments provided by `nicematrix` as explained previously.

In the same way, if the option `renew-matrix` is used, the environment `{matrix}` of `amsmath` is redefined.

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

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

Declaration of the package and packages loaded

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

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

<@@=nicematrix>

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

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

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

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

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

We load some packages.

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

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

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

Technical definitions

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

24 \bool_new:N \c_@@_booktabs_loaded_bool
25 \bool_new:N \c_@@_enumitem_loaded_bool
26 \bool_new:N \c_@@_tikz_loaded_bool
27 \AtBeginDocument
28   {
29     \ifpackageloaded { booktabs }
30       { \bool_set_true:N \c_@@_booktabs_loaded_bool }
31   }
```

```

31 { }
32 \@ifpackageloaded { enumitem }
33 { \bool_set_true:N \c_@@_enumitem_loaded_bool }
34 { }
35 \@ifpackageloaded { tikz }
36 {

```

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

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

```

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

```

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

```

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

```

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

```

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

```

This definition is a variant of the standard definition of `\ddots`.

In the `aux` file, we will have the references of the PGF/Tikz nodes created by `nicematrix`. However, when `booktabs` is used, some nodes (more precisely, some `row` nodes) will be defined twice because their position will be modified. In order to avoid an error message in this case, we will redefine `\pgfutil@check@rerun` in the `aux` file.

```

67 \AtBeginDocument
68 {
69 \@ifpackageloaded { booktabs }
70 { \iow_now:Nn \mainaux \nicematrix@redefine@check@rerun }
71 { }

```

```

72 }
73 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
74 {
75   \cs_set_eq:NN \@@_old_pgfulil@check@rerun \pgfulil@check@rerun

```

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

```

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

```

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

```

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

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. We will use it to store the instruction of color for the rules even if `colortbl` is not loaded. Idem for

```

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

```

Idem for `\CT@drs@`.

```

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

```

We have to redefine `\cline` for several reasons. The command `\@@_cline` will be linked to `\cline` in the beginning of `{NiceArrayWithDelims}`. The following commands must *not* be protected.

```

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

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`³³

```
123   \skip_horizontal:N \c_zero_dim
124 }
```

Our `\everycr` has been modified. In particular, the creation of the `row` node is in the `\everycr` (maybe we should put it with the incrementation of `\c@iRow`). Since the following `\cr` correspond to a “false row”, we have to nullify `\everycr`.

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

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

```
129 \cs_set:Npn @@_cline
```

We have to act in a fully expandable way since there may be `\noalign` (in the `\multispan`) to detect. That’s why we use `@@_cline_i:en`.

```
130 { @@_cline_i:en \l_@@_first_col_int }
```

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

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

Now, `#1` is the number of the current column and we have to draw a line from the column `#2` to the column `#3` (both included).

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

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

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

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

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

The following command is a small shortcut.

```
149 \cs_new:Npn @@_math_toggle_token:
150 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }
```

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

```
157 \cs_new:Npn @@_tab_or_array_colsep:
```

³³See question 99041 on TeX StackExchange.

```
158 { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
```

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`.

```
159 \bool_new:N \c_@@_siunitx_loaded_bool
160 \AtBeginDocument
161 {
162   \ifpackageloaded { siunitx }
163     { \bool_set_true:N \c_@@_siunitx_loaded_bool }
164     { }
165 }
```

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

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

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

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

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

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

```
166 \cs_set_protected:Npn \@@_adapt_S_column:
167 {
168   \bool_if:NT \c_@@_siunitx_loaded_bool
169   {
170     \group_begin:
171     \@temptokena = { }
```

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

```
172   \cs_set_eq:NN \NC@find \prg_do_nothing:
173   \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).

```

174     \tl_gset:NV \g_tmpa_tl \@temptokena
175     \group_end:
176     \tl_new:N \c_@@_table_collect_begin_tl
177     \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
178     \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
179     \tl_new:N \c_@@_table_print_tl
180     \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).

```

181     \cs_gset_eq:NN \@_adapt_S_column: \prg_do_nothing:
182 }
183 }

```

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

```

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

```

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

```

197         \@_true_c:
198         < { \c_@@_table_print_tl \@_end_Cell: }
199     }
200     \NC@find
201 }
202 }
203 }
204 }

```

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

```

205 \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }

```

If the final user uses nicematrix, PGF/Tikz will write instruction \pgfsyspdfmark in the aux file. If he changes its mind and no longer loads nicematrix, an error may occur at the next compilation because of remanent instructions \pgfsyspdfmark in the aux file. With the following code, we avoid that situation.

```

206 \cs_new_protected:Npn \@_provide_pgfsyspdfmark:
207 {
208     \iow_now:Nn \@mainaux
209     {
210         \ExplSyntaxOn
211         \cs_if_free:NT \pgfsyspdfmark
212         { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
213         \ExplSyntaxOff
214     }
215     \cs_gset_eq:NN \@_provide_pgfsyspdfmark: \prg_do_nothing:
216 }

```

Parameters

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

```
217 \bool_new:N \c_@@_define_L_C_R_bool
218 \cs_new_protected:Npn \@@_define_L_C_R:
219 {
220   \newcolumntype L l
221   \newcolumntype C c
222   \newcolumntype R r
223 }
```

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.

```
224 \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).

```
225 \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.

```
226 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
227 { \int_use:N \g_@@_env_int }
```

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

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

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

```
230 \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).

```
231 \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`.

```
232 \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.

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

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

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

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

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


If the user uses `{NiceTabular*}`, the width of the tabular (in the first argument of the environment `{NiceTabular*}`) will be stored in the following dimension.

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

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

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

```
238 \cs_new_protected:Npn \@@_test_if_math_mode:
239 {
240   \if_mode_math: \else:
241     \@@_fatal:n { Outside~math~mode }
242   \fi:
243 }
```

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

```
244 \colorlet { nicematrix-last-col } { . }
245 \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`).

```
246 \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*.

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

The following command will be able to reconstruct the full name of the current command or environment (despite its name which contains `env`). This command must *not* be protected since it will be used in error messages.

```
248 \cs_new:Npn \@@_full_name_env:
249 {
250   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
251     { command \space \c_backslash_str \g_@@_name_env_str }
252     { environment \space \{ \g_@@_name_env_str \} }
253 }
```

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`).

```
254 \tl_new:N \g_nicematrix_code_after_tl
```

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

```
255 \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.

```
256 \int_new:N \l_@@_old_iRow_int
257 \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.

```
258 \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).

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

`\l_@@_code_before_tl` may contain two types of informations:

- A `code-before` written in the `aux` file by a previous run. When the `aux` file is read, this `code-before` is stored in `\g_@@_code_before_i_tl` (where i is the number of the environment) and, at the beginning of the environment, it will be put in `\l_@@_code_before_tl`.
- The final user can explicitly add material in `\l_@@_code_before_tl` by using the key `code-before`.

```
260 \tl_new:N \l_@@_code_before_tl
```

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

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

```
262 \dim_new:N \l_@@_x_initial_dim
```

```
263 \dim_new:N \l_@@_y_initial_dim
```

```
264 \dim_new:N \l_@@_x_final_dim
```

```
265 \dim_new:N \l_@@_y_final_dim
```

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

```
266 \dim_zero_new:N \l_tmpc_dim
```

```
267 \dim_zero_new:N \l_tmpd_dim
```

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

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

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

```
269 \dim_new:N \@@_old_arraycolsep_dim
```

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

```
270 \dim_new:N \g_@@_width_last_col_dim
```

```
271 \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.

```
272 \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}`.

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

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

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

```
274 \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).

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble, of course and without the potential exterior columns).

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

Used for the color of the blocks.

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

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.

```
277 \int_new:N \l_@@_first_row_int
278 \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.

```
279 \int_new:N \l_@@_first_col_int
280 \int_set:Nn \l_@@_first_col_int 1
```

• Last row

The counter `\l_@@_last_row_int` is the number of the potential “last row”, as specified by the key `last-row`. A value of `-2` means that there is no “last row”. A value of `-1` means that there is a “last row” but we don't know the number of that row (the key `last-row` has been used without value and the actual value has not still been read in the `aux` file).

```
281 \int_new:N \l_@@_last_row_int
282 \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”.³⁴

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

Idem for `\l_@@_last_col_without_value_bool`

```
284 \bool_new:N \l_@@_last_col_without_value_bool
```

³⁴We 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.

- **Last column**

For the potential “last column”, we use an integer. A value of -2 means that there is no last column. A value of -1 means that we are in an environment without preamble (e.g. `{bNiceMatrix}`) and there is a last column but we don’t know its value because the user has used the option `last-col` without value. A value of 0 means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`): in this case, the key was necessary without argument.

```
285 \int_new:N \l_@@_last_col_int
286 \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.

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

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

The command `\tablarnote`

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

```
288 \newcounter { tablarnote }
```

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

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

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

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

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

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

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

```
292 \cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }
```

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

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

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

```
294 \cs_set:Npn \thetablarnote { { \@@_notes_style:n { tablarnote } } }
```

The tabular notes will be available for the final user only when `enumitem` is loaded. Indeed, the tabular notes will be composed at the end of the array with a list customized by `enumitem` (a list `tabularnotes` in the general case and a list `tabularnotes*` if the key `para` is in force). However, we can test whether `enumitem` has been loaded only at the beginning of the document (we want to allow the user to load `enumitem` after `nicematrix`).

```

295 \AtBeginDocument
296 {
297   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
298   {
299     \NewDocumentCommand \tabularnote { m }
300     { \@@_error:n { enumitem~not~loaded } }
301   }
302   {

```

The type of list `tabularnotes` will be used to format the tabular notes at the end of the array in the general case and `tabularnotes*` will be used if the key `para` is in force.

```

303   \newlist { tabularnotes } { enumerate } { 1 }
304   \setlist [ tabularnotes ]
305   {
306     noitemsep , leftmargin = * , align = left , labelsep = 0pt ,
307     label =
308       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
309   }
310   \newlist { tabularnotes* } { enumerate* } { 1 }
311   \setlist [ tabularnotes* ]
312   {
313     afterlabel = \nobreak ,
314     itemjoin = \quad ,
315     label =
316       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
317   }

```

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

Unfortunately, if the package `caption` is loaded, the command `\caption` evaluates its argument twice and since it is not aware (of course) of `\tabularnote`, the command `\tabularnote` is, in fact, not usable in `\caption` when `caption` is loaded.³⁵

```

318   \NewDocumentCommand \tabularnote { m }
319   {
320     \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
321     { \@@_error:n { tabularnote~forbidden } }
322     {

```

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

```

323       \int_incr:N \l_@@_number_of_notes_int

```

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

```

324       \seq_gput_right:Nx \g_@@_tabularnotes_seq { #1 }
325       \peek_meaning:NF \tabularnote
326       {

```

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

```

327       \hbox_set:Nn \l_tmpa_box
328       {

```

³⁵We should try to find a solution to that problem.

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

```

329         \@@_notes_label_in_tabular:n
330         {
331             \stepcounter { tabularnote }
332             \@@_notes_style:n { tabularnote }
333             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
334             {
335                 ,
336                 \stepcounter { tabularnote }
337                 \@@_notes_style:n { tabularnote }
338             }
339         }
340     }

```

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

```

341         \addtocounter { tabularnote } { -1 }
342         \refstepcounter { tabularnote }
343         \int_zero:N \l_@@_number_of_notes_int
344         \hbox_overlap_right:n { \box_use:N \l_tmpa_box }

```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array?`) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```

345         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
346     }
347 }
348 }
349 }
350 }

```

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.

```

351 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
352 {
353     \begin { pgfscope }
354     \pgfset
355     {
356         outer~sep = \c_zero_dim ,
357         inner~sep = \c_zero_dim ,
358         minimum~size = \c_zero_dim
359     }
360     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
361     \pgfnode
362     { rectangle }
363     { center }
364     {
365         \vbox_to_ht:nn
366         { \dim_abs:n { #5 - #3 } }
367         {
368             \vfill
369             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
370         }
371     }
372     { #1 }
373     { }

```

```

374 \end { pgfscope }
375 }

```

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.

```

376 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
377 {
378   \begin { pgfscope }
379   \pgfset
380   {
381     outer~sep = \c_zero_dim ,
382     inner~sep = \c_zero_dim ,
383     minimum~size = \c_zero_dim
384   }
385   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
386   \pgfpointdiff { #3 } { #2 }
387   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
388   \pgfnode
389   { rectangle }
390   { center }
391   {
392     \vbox_to_ht:nn
393     { \dim_abs:n \l_tmpb_dim }
394     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
395   }
396   { #1 }
397   { }
398   \end { pgfscope }
399 }

```

The options

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

```

400 \bool_new:N \l_@@_colortbl_like_bool

```

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

```

401 \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`).

```

402 \dim_new:N \l_@@_cell_space_top_limit_dim
403 \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.

```

404 \dim_new:N \l_@@_inter_dots_dim
405 \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).

```

406 \dim_new:N \l_@@_xdots_shorten_dim
407 \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.

```
408 \dim_new:N \l_@@_radius_dim
409 \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.

```
410 \tl_new:N \l_@@_xdots_line_style_tl
411 \tl_const:Nn \c_@@_standard_tl { standard }
412 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
413 \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).

```
414 \str_new:N \l_@@_baseline_str
415 \tl_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`).

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

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

```
417 \bool_new:N \l_@@_parallelize_diags_bool
418 \bool_set_true:N \l_@@_parallelize_diags_bool
```

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

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

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

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

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

```
421 \clist_new:N \l_@@_except_corners_clist
422 \dim_new:N \l_@@_notes_above_space_dim
423 \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm }
```

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

```
424 \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).

```
425 \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.

```
426 \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”.

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

```
428 \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`).

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

```
430 \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`.

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

```
432 \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.

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

```
434 \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 “:”.

```
435 \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.

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

```
437 \keys_define:nn { NiceMatrix / xdots }
438 {
439   line-style .code:n =
440   {
441     \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.

```
442     { \cs_if_exist_p:N \tikzpicture }
443     { \str_if_eq_p:nn { #1 } { standard } }
444     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
445     { \@@_error:n { bad-option~for~line-style } }
446   } ,
447   line-style .value_required:n = true ,
448   color .tl_set:N = \l_@@_xdots_color_tl ,
449   color .value_required:n = true ,
450   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
451   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 `_`.

```

452   down .tl_set:N = \l_@@_xdots_down_tl ,
453   up .tl_set:N = \l_@@_xdots_up_tl ,
454   unknown .code:n = \@@_error:n { Unknown-option-for-~xdots }
455 }

```

The following keys are for the tabular notes (specified by the command `\tabularnote` inside `{NiceTabular}` and `{NiceArray}`).

```

456 \keys_define:nn { NiceMatrix / rules }
457 {
458   color .tl_set:N = \l_@@_rules_color_tl ,
459   color .value_required:n = true ,
460   width .dim_set:N = \arrayrulewidth ,
461   width .value_required:n = true
462 }

```

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

```

463 \keys_define:nn { NiceMatrix / Global }
464 {
465   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
466   standard-cline .default:n = true ,
467   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
468   cell-space-top-limit .value_required:n = true ,
469   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
470   cell-space-bottom-limit .value_required:n = true ,
471   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
472   max-delimiter-width .bool_set:N = \l_@@_max_delimiter_width_bool ,
473   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
474   light-syntax .default:n = true ,
475   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
476   end-of-row .value_required:n = true ,
477   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
478   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
479   last-row .int_set:N = \l_@@_last_row_int ,
480   last-row .default:n = -1 ,
481   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
482   code-for-first-col .value_required:n = true ,
483   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
484   code-for-last-col .value_required:n = true ,
485   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
486   code-for-first-row .value_required:n = true ,
487   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
488   code-for-last-row .value_required:n = true ,
489   hlines .bool_set:N = \l_@@_hlines_bool ,
490   vlines .bool_set:N = \l_@@_vlines_bool ,
491   hvlines .code:n =
492   {
493     \bool_set_true:N \l_@@_vlines_bool
494     \bool_set_true:N \l_@@_hlines_bool
495   } ,
496   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.

```

497   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
498   renew-dots .value_forbidden:n = true ,
499   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,

```

```

500 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
501 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
502 create-extra-nodes .meta:n =
503   { create-medium-nodes , create-large-nodes } ,
504 left-margin .dim_set:N = \l_@@_left_margin_dim ,
505 left-margin .default:n = \arraycolsep ,
506 right-margin .dim_set:N = \l_@@_right_margin_dim ,
507 right-margin .default:n = \arraycolsep ,
508 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
509 margin .default:n = \arraycolsep ,
510 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
511 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
512 extra-margin .meta:n =
513   { extra-left-margin = #1 , extra-right-margin = #1 } ,
514 extra-margin .value_required:n = true ,
515 }

```

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

```

516 \keys_define:nn { NiceMatrix / Env }
517 {
518   except-corners .clist_set:N = \l_@@_except_corners_clist ,
519   except-corners .default:n = { NW , SW , NE , SE } ,
520   hvlines-except-corners .code:n =
521     {
522       \clist_set:Nn \l_@@_except_corners_clist { #1 }
523       \bool_set_true:N \l_@@_vlines_bool
524       \bool_set_true:N \l_@@_hlines_bool
525     } ,
526   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
527   code-before .code:n =
528     {
529       \tl_if_empty:nF { #1 }
530       {
531         \tl_put_right:Nn \l_@@_code_before_tl { #1 }
532         \bool_set_true:N \l_@@_code_before_bool
533       }
534     } ,

```

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

```

535 c .code:n = \tl_set:Nn \l_@@_baseline_str c ,
536 t .code:n = \tl_set:Nn \l_@@_baseline_str t ,
537 b .code:n = \tl_set:Nn \l_@@_baseline_str b ,
538 baseline .tl_set:N = \l_@@_baseline_str ,
539 baseline .value_required:n = true ,
540 columns-width .code:n =
541   \tl_if_eq:nnTF { #1 } { auto }
542     { \bool_set_true:N \l_@@_auto_columns_width_bool }
543     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
544 columns-width .value_required:n = true ,
545 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.

```

546 \legacy_if:nF { measuring@ }
547 {
548   \str_set:Nn \l_tmpa_str { #1 }
549   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
550     { \@@_error:nn { Duplicate-name } { #1 } }
551     { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
552   \str_set_eq:NN \l_@@_name_str \l_tmpa_str

```

```

553     } ,
554     name .value_required:n = true ,
555     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
556     code-after .value_required:n = true ,
557     colortbl-like .code:n =
558       \bool_set_true:N \l_@@_colortbl_like_bool
559       \bool_set_true:N \l_@@_code_before_bool ,
560     colortbl-like .value_forbidden:n = true
561   }
562   \keys_define:nn { NiceMatrix / notes }
563   {
564     para .bool_set:N = \l_@@_notes_para_bool ,
565     para .default:n = true ,
566     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
567     code-before .value_required:n = true ,
568     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
569     code-after .value_required:n = true ,
570     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
571     bottomrule .default:n = true ,
572     style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
573     style .value_required:n = true ,
574     label-in-tabular .code:n =
575       \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
576     label-in-tabular .value_required:n = true ,
577     label-in-list .code:n =
578       \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
579     label-in-list .value_required:n = true ,
580     enumitem-keys .code:n =
581       {
582         \bool_if:NTF \c_@@_in_preamble_bool
583         {
584           \AtBeginDocument
585           {
586             \bool_if:NT \c_@@_enumitem_loaded_bool
587             { \setlist* [ tabularnotes ] { #1 } }
588           }
589         }
590         {
591           \bool_if:NT \c_@@_enumitem_loaded_bool
592           { \setlist* [ tabularnotes ] { #1 } }
593         }
594       } ,
595     enumitem-keys .value_required:n = true ,
596     enumitem-keys-para .code:n =
597       {
598         \bool_if:NTF \c_@@_in_preamble_bool
599         {
600           \AtBeginDocument
601           {
602             \bool_if:NT \c_@@_enumitem_loaded_bool
603             { \setlist* [ tabularnotes* ] { #1 } }
604           }
605         }
606         {
607           \bool_if:NT \c_@@_enumitem_loaded_bool
608           { \setlist* [ tabularnotes* ] { #1 } }
609         }
610       } ,
611     enumitem-keys-para .value_required:n = true ,
612     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
613   }

```

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

environments).

```

614 \keys_define:nn { NiceMatrix }
615 {
616   NiceMatrixOptions .inherit:n =
617     { NiceMatrix / Global } ,
618   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
619   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
620   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
621   NiceMatrix .inherit:n =
622     {
623       NiceMatrix / Global ,
624       NiceMatrix / Env ,
625     } ,
626   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
627   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
628   NiceTabular .inherit:n =
629     {
630       NiceMatrix / Global ,
631       NiceMatrix / Env
632     } ,
633   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
634   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
635   NiceArray .inherit:n =
636     {
637       NiceMatrix / Global ,
638       NiceMatrix / Env ,
639     } ,
640   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
641   NiceArray / rules .inherit:n = NiceMatrix / rules ,
642   pNiceArray .inherit:n =
643     {
644       NiceMatrix / Global ,
645       NiceMatrix / Env ,
646     } ,
647   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
648   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
649 }

```

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

```

650 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
651 {
652   last-col .code:n = \tl_if_empty:nF { #1 }
653     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
654     \int_zero:N \l_@@_last_col_int ,
655   small .bool_set:N = \l_@@_small_bool ,
656   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.

```

657   renew-matrix .code:n = \@@_renew_matrix: ,
658   renew-matrix .value_forbidden:n = true ,
659   transparent .meta:n = { renew-dots , renew-matrix } ,
660   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}`.

```

661   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.

```

662 columns-width .code:n =
663   \tl_if_eq:nnTF { #1 } { auto }
664   { \@@_error:n { Option~auto~for~columns~width } }
665   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

666 allow-duplicate-names .code:n =
667   \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
668 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`).

```

669 letter-for-dotted-lines .code:n =
670   {
671     \tl_if_single_token:nTF { #1 }
672     { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
673     { \@@_error:n { Bad~value~for~letter~for~dotted~lines } }
674   } ,
675 letter-for-dotted-lines .value_required:n = true ,
676 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
677 notes .value_required:n = true ,
678 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
679 }
680 \str_new:N \l_@@_letter_for_dotted_lines_str
681 \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.

```

682 \NewDocumentCommand \NiceMatrixOptions { m }
683 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

684 \keys_define:nn { NiceMatrix / NiceMatrix }
685 {
686   last-col .code:n = \tl_if_empty:nTF {#1}
687   {
688     \bool_set_true:N \l_@@_last_col_without_value_bool
689     \int_set:Nn \l_@@_last_col_int { -1 }
690   }
691   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
692   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
693   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
694   small .bool_set:N = \l_@@_small_bool ,
695   small .value_forbidden:n = true ,
696   unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
697 }

```

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

```

698 \keys_define:nn { NiceMatrix / NiceArray }
699 {

```

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.

```

700   small .bool_set:N = \l_@@_small_bool ,
701   small .value_forbidden:n = true ,
702   last-col .code:n = \tl_if_empty:nF { #1 }
703       { \@@_error:n { last-col~non-empty~for~NiceArray } }
704       \int_zero:N \l_@@_last_col_int ,
705   notes / para .bool_set:N = \l_@@_notes_para_bool ,
706   notes / para .default:n = true ,
707   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
708   notes / bottomrule .default:n = true ,
709   unknown .code:n = \@@_error:n { Unknown~option~for~NiceArray }
710 }

711 \keys_define:nn { NiceMatrix / pNiceArray }
712 {
713   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
714   last-col .code:n = \tl_if_empty:nF {#1}
715       { \@@_error:n { last-col~non-empty~for~NiceArray } }
716       \int_zero:N \l_@@_last_col_int ,
717   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
718   small .bool_set:N = \l_@@_small_bool ,
719   small .value_forbidden:n = true ,
720   unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
721 }

```

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

```

722 \keys_define:nn { NiceMatrix / NiceTabular }
723 {
724   notes / para .bool_set:N = \l_@@_notes_para_bool ,
725   notes / para .default:n = true ,
726   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
727   notes / bottomrule .default:n = true ,
728   last-col .code:n = \tl_if_empty:nF {#1}
729       { \@@_error:n { last-col~non-empty~for~NiceArray } }
730       \int_zero:N \l_@@_last_col_int ,
731   unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular }
732 }

```

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}`).

```

733 \cs_new_protected:Npn \@@_Cell:
734 {

```

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

```

735   \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.

```

736   \int_compare:nNnT \c@jCol = 1
737   { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
738   \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).

```

739   \hbox_set:Nw \l_@@_cell_box
740   \bool_if:NF \l_@@_NiceTabular_bool
741   {
742     \c_math_toggle_token
743     \bool_if:NT \l_@@_small_bool \scriptstyle
744   }

```

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.

```

745   \int_compare:nNnTF \c@iRow = 0
746   {
747     \int_compare:nNnT \c@jCol > 0
748     {
749       \l_@@_code_for_first_row_tl
750       \xglobal \colorlet { nicematrix-first-row } { . }
751     }
752   }
753   {
754     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
755     {
756       \l_@@_code_for_last_row_tl
757       \xglobal \colorlet { nicematrix-last-row } { . }
758     }
759   }
760 }

```

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.

```

761 \cs_new_protected:Npn \@@_begin_of_row:
762 {
763   \int_gincr:N \c@iRow
764   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
765   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
766   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
767   \pgfpicture
768   \pgfrememberpicturepositiononpagetrue
769   \pgfcoordinate
770   { \@@_env: - row - \int_use:N \c@iRow - base }
771   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
772   \str_if_empty:NF \l_@@_name_str
773   {
774     \pgfnodealias
775     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
776     { \@@_env: - row - \int_use:N \c@iRow - base }
777   }
778   \endpgfpicture
779 }

```

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command.

```

780 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
781 {
782   \int_compare:nNnTF \c@iRow = 0
783   {
784     \dim_gset:Nn \g_@@_dp_row_zero_dim

```



```

785     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
786   \dim_gset:Nn \g_@@_ht_row_zero_dim
787     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
788 }
789 {
790   \int_compare:nNnT \c@iRow = 1
791   {
792     \dim_gset:Nn \g_@@_ht_row_one_dim
793       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
794   }
795 }
796 }
797 \cs_new_protected:Npn \@@_end_Cell:
798 {
799   \@@_math_toggle_token:
800   \hbox_set_end:
801   \box_set_ht:Nn \l_@@_cell_box
802     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
803   \box_set_dp:Nn \l_@@_cell_box
804     { \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”).

```

805   \dim_gset:Nn \g_@@_max_cell_width_dim
806     { \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”.

```

807   \@@_update_for_first_and_last_row:

```

If the cell is empty, or may be considered as if, we must not create the PGF node, for two reasons:

- it’s a waste of time since such a node would be rather pointless;
- we test the existence of these nodes in order to determine whether a cell is empty when we search the extremities of a dotted line.

However, it’s very difficult to determine whether a cell is empty. As of now, we use the following technic:

- if the width of the box `\l_@@_cell_box` (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a `\rlap`, a `\llap` or a `\mathclap` of `mathtools`).
- the cells with a command `\Ldots` or `\Cdots`, `\Vdots`, etc., should also be considered as empty; if `nullify-dots` is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of `code-after`); however, if `nullify-dots` is not in force, a phantom of `\ldots`, `\cdots`, `\vdots` is inserted and its width is not equal to zero; that’s why these commands raise a boolean `\g_@@_empty_cell_bool` and we begin by testing this boolean.

```

808   \bool_if:NTF \g_@@_empty_cell_bool
809     { \box_use_drop:N \l_@@_cell_box }
810   {
811     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
812       \@@_node_for_the_cell:
813       { \box_use_drop:N \l_@@_cell_box }
814   }
815   \bool_gset_false:N \g_@@_empty_cell_bool
816 }

```

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

```

817 \cs_new_protected:Npn \@@_node_for_the_cell:
818 {
819   \pgfpicture

```

```

820 \pgfsetbaseline \c_zero_dim
821 \pgfrememberpicturepositiononpagetrue
822 \pgfset
823 {
824     inner-sep = \c_zero_dim ,
825     minimum-width = \c_zero_dim
826 }
827 \pgfnode
828 { rectangle }
829 { base }
830 { \box_use_drop:N \l_@@_cell_box }
831 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
832 { }
833 \str_if_empty:NF \l_@@_name_str
834 {
835     \pgfnodealias
836     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
837     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
838 }
839 \endpgfpicture
840 }

```

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}

```

```

841 \cs_new_protected:Npn \@@_instruction_of_type:nn #1 #2
842 {

```

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).

```

843 \tl_gput_right:cx
844 { g_@@_ #1 _ lines _ tl }
845 {
846     \use:c { @@ _ draw _ #1 : nnn }
847     { \int_use:N \c@iRow }
848     { \int_use:N \c@jCol }
849     { \exp_not:n { #2 } }
850 }
851 }

```

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`.

```

852 \cs_new_protected:Npn \@@_revtex_array:
853 {
854     \cs_set_eq:NN \@acol1 \@arrayacol
855     \cs_set_eq:NN \@acolr \@arrayacol
856     \cs_set_eq:NN \@acol \@arrayacol

```

```

857 \cs_set:Npn \@halignto { }
858 \@array@array
859 }
860 \cs_new_protected:Npn \@_array:
861 {
862   \bool_if:NTF \c_@@_revtex_bool
863     \@_revtex_array:
864     {
865       \bool_if:NTF \l_@@_NiceTabular_bool
866         { \dim_set_eq:NN \col@sep \tabcolsep }
867         { \dim_set_eq:NN \col@sep \arraycolsep }
868       \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
869         { \cs_set:Npn \@halignto { } }
870         { \cs_set:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

871 \@tabarray
872 }

```

\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.

```

873 [ \str_if_eq:VnTF \l_@@_baseline_str c c t ]
874 }

```

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.

```

875 \cs_set_eq:NN \@_old_ialign: \ialign

```

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

```

876 \cs_new_protected:Npn \@_create_row_node:
877 {

```

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

```

878 \hbox
879 {
880   \bool_if:NT \l_@@_code_before_bool
881     {
882       \vtop
883       {
884         \skip_vertical:N 0.5\arrayrulewidth
885         \pgfsys@markposition { \@_env: - row - \@_succ:n \c@iRow }
886         \skip_vertical:N -0.5\arrayrulewidth
887       }
888     }
889   \pgfpicture
890   \pgfrememberpicturepositiononpagetrue
891   \pgfcoordinate { \@_env: - row - \@_succ:n \c@iRow }
892   { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
893   \str_if_empty:NF \l_@@_name_str
894     {
895       \pgfnodealias
896       { \l_@@_name_str - row - \int_use:N \c@iRow }
897       { \@_env: - row - \int_use:N \c@iRow }
898     }
899   \endpgfpicture
900 }
901 }

```

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

```

902 \cs_new:Npn \@_everycr: { \noalign { \@_everycr_i: } }

```

```

903 \cs_new_protected:Npn \@@_everycr_i:
904 {
905     \int_gzero:N \c@jCol
906     \bool_if:NF \g_@@_row_of_col_done_bool
907     {
908         \@@_create_row_node:

```

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

```

909         \bool_if:NT \l_@@_hlines_bool
910     }

```

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.

```

911         \int_compare:nNnT \c@iRow > { -1 }
912         {
913             \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@`.

```

914             { \hrule height \arrayrulewidth width \c_zero_dim }
915         }
916     }
917 }
918 }

```

The command `\@@_newcolumntype` is the command `\newcolumntype` of `array` without the warnings for redefinitions of columns types (we will use it to redefine the columns types `w` and `W`).

```

919 \cs_set_protected:Npn \@@_newcolumntype #1
920 {
921     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
922     \peek_meaning:NTF [
923         { \newcol@ #1 }
924         { \newcol@ #1 [ 0 ] }
925     }

```

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

```

926 \cs_set_protected:Npn \@@_renew_dots:
927 {
928     \cs_set_eq:NN \ldots \@@_Ldots
929     \cs_set_eq:NN \cdots \@@_Cdots
930     \cs_set_eq:NN \vdots \@@_Vdots
931     \cs_set_eq:NN \ddots \@@_Ddots
932     \cs_set_eq:NN \iddots \@@_Iddots
933     \cs_set_eq:NN \dots \@@_Ldots
934     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
935 }

```

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

```

936 \cs_new_protected:Npn \@@_colortbl_like:
937 {
938     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
939     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
940     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
941 }

```

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

```

942 \cs_new_protected:Npn \@@_pre_array:
943 {

```

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 ³⁶.

```

944 \bool_if:NT \c_@@_booktabs_loaded_bool
945   { \tl_put_left:Nn \@BTnormal \@_create_row_node: }
946 \box_clear_new:N \l_@@_cell_box
947 \cs_if_exist:NT \theiRow
948   { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
949 \int_gzero_new:N \c@iRow
950 \cs_if_exist:NT \thejCol
951   { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
952 \int_gzero_new:N \c@jCol
953 \normalbaselines

```

If the option `small` is used, we have to do some tuning. In particular, we change the value of `\arraystretch` (this parameter is used in the construction of `\@arstrutbox` in the beginning of `{array}`).

```

954 \bool_if:NT \l_@@_small_bool
955   {
956     \cs_set:Npn \arraystretch { 0.47 }
957     \dim_set:Nn \arraycolsep { 1.45 pt }
958   }

```

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`.

```

959 \cs_set:Npn \ialign
960   {
961     \bool_if:NTF \c_@@_colortbl_loaded_bool
962       {
963         \CT@everycr
964         {
965           \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
966           \@_everycr:
967         }
968       }
969     { \everycr { \@_everycr: } }
970     \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`³⁷ 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`.

```

971 \dim_gzero_new:N \g_@@_dp_row_zero_dim
972 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
973 \dim_gzero_new:N \g_@@_ht_row_zero_dim
974 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
975 \dim_gzero_new:N \g_@@_ht_row_one_dim
976 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
977 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
978 \dim_gzero_new:N \g_@@_ht_last_row_dim
979 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }

```

³⁶cf. `\nicematrix@redefine@check@rerun`

³⁷The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

980 \dim_gzero_new:N \g_@@_dp_last_row_dim
981 \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`.

```

982 \cs_set_eq:NN \ialign \@@_old_ialign:
983 \halign
984 }

```

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).

```

985 \cs_set_eq:NN \@@_old_ldots \ldots
986 \cs_set_eq:NN \@@_old_cdots \cdots
987 \cs_set_eq:NN \@@_old_vdots \vdots
988 \cs_set_eq:NN \@@_old_ddots \ddots
989 \cs_set_eq:NN \@@_old_iddots \iddots
990 \bool_if:NTF \l_@@_standard_cline_bool
991 { \cs_set_eq:NN \cline \@@_standard_cline }
992 { \cs_set_eq:NN \cline \@@_cline }
993 \cs_set_eq:NN \Ldots \@@_Ldots
994 \cs_set_eq:NN \Cdots \@@_Cdots
995 \cs_set_eq:NN \Vdots \@@_Vdots
996 \cs_set_eq:NN \Ddots \@@_Ddots
997 \cs_set_eq:NN \Iddots \@@_Iddots
998 \cs_set_eq:NN \hdottedline \@@_hdottedline:
999 \cs_set_eq:NN \Hline \@@_Hline:
1000 \cs_set_eq:NN \Hspace \@@_Hspace:
1001 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1002 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1003 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1004 \cs_set_eq:NN \Block \@@_Block:
1005 \cs_set_eq:NN \rotate \@@_rotate:
1006 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1007 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1008 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:n
1009 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1010 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1011 \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:

```

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

```

1012 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1013 \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).

```

1014 \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 of rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1015 \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.

```

1016 \int_gzero_new:N \g_@@_col_total_int
1017 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1018 \@@_renew_NC@rewrite@S:
1019 \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.

```

1020 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1021 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1022 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1023 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1024 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1025 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1026 \tl_gclear_new:N \g_nicematrix_code_before_tl
1027 }

```

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

The environment `{NiceArrayWithDelims}`

```

1028 \NewDocumentEnvironment { NiceArrayWithDelims } { m m 0 { } m ! 0 { } }
1029 {
1030   \@@_provide_pgfsyspdfmark:
1031   \bool_if:NT \c_@@_footnote_bool \savenotes

```

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

```

1032 \bgroup
1033 \tl_set:Nn \l_@@_left_delim_tl { #1 }
1034 \tl_set:Nn \l_@@_right_delim_tl { #2 }
1035 \bool_gset_false:N \g_@@_row_of_col_done_bool
1036 \str_if_empty:NT \g_@@_name_env_str
1037   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1038 \@@_adapt_S_column:
1039 \bool_if:NTF \l_@@_NiceTabular_bool
1040   \mode_leave_vertical:
1041   \@@_test_if_math_mode:
1042   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1043   \bool_set_true:N \l_@@_in_env_bool

```

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 beginning of arrays done by `colortbl`. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```

1044 \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).

```

1045 \cs_if_exist:NT \tikz@library@external@loaded
1046 {
1047   \tikzset { external / export = false }
1048   \cs_if_exist:NT \ifstandalone
1049     { \tikzset { external / optimize = false } }
1050 }

```

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

```

1051 \int_gincr:N \g_@@_env_int
1052 \bool_if:NF \l_@@_block_auto_columns_width_bool
1053   { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1054 \seq_gclear:N \g_@@_blocks_seq
1055 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

³⁸e.g. `\color[rgb]{0.5,0.5,0}`

```
1056 \seq_gclear:N \g_@@_pos_of_xdots_seq
```

In fact, the sequence `\g_@@_pos_of_blocks_seq` will also contain the positions of the cells with a `\diagbox`.

```
1057 \tl_if_exist:cT { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
1058 {
1059   \bool_set_true:N \l_@@_code_before_bool
1060   \exp_args:NNv \tl_put_right:Nn \l_@@_code_before_tl
1061   { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
1062 }
```

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`.

```
1063 \bool_if:NTF \l_@@_NiceArray_bool
1064 { \keys_set:nn { NiceMatrix / NiceArray } }
1065 { \keys_set:nn { NiceMatrix / pNiceArray } }
1066 { #3 , #5 }
```

```
1067 \tl_if_empty:NF \l_@@_rules_color_tl
1068 { \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.

```
1069 \bool_if:NT \l_@@_code_before_bool
1070 {
1071   \seq_if_exist:cT { @@_size_ \int_use:N \g_@@_env_int _ seq }
1072   {
```

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).

```
1073   \int_zero_new:N \c@iRow
1074   \int_set:Nn \c@iRow
1075   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
1076   \int_zero_new:N \c@jCol
1077   \int_set:Nn \c@jCol
1078   { \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.

```
1079   \int_compare:nNnF \l_@@_last_row_int = { -2 }
1080   { \int_decr:N \c@iRow }
1081   \int_compare:nNnF \l_@@_last_col_int = { -2 }
1082   { \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.

```
1083   \pgfsys@markposition { \@@_env: - position }
1084   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1085   \pgfpicture
```

First, the creation of the `row` nodes.

```
1086   \int_step_inline:nnn
1087   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 1 }
1088   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 + 1 }
1089   {
1090     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1091     \pgfcoordinate { \@@_env: - row - ##1 }
1092     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1093   }
```

Now, the creation of the `col` nodes.

```
1094   \int_step_inline:nnn
```



```

1095 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
1096 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 + 1 }
1097 {
1098   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1099   \pgfcoordinate { \@@_env: - col - ##1 }
1100   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1101 }
1102 \endpgfpicture
1103 \group_begin:
1104   \bool_if:NT \c_@@_tikz_loaded_bool
1105   {
1106     \tikzset
1107     {
1108       every-picture / .style =
1109       { overlay , name-prefix = \@@_env: - }
1110     }
1111   }
1112   \cs_set_eq:NN \cellcolor \@@_cellcolor
1113   \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1114   \cs_set_eq:NN \rowcolor \@@_rowcolor
1115   \cs_set_eq:NN \rowcolors \@@_rowcolors
1116   \cs_set_eq:NN \columncolor \@@_columncolor
1117   \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.

```

1118   \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1119   \l_@@_code_before_tl
1120   \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1121 \group_end:
1122 }
1123 }

```

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).

```

1124 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1125 {
1126   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1127   {
1128     \dim_gset:Nn \g_@@_ht_last_row_dim
1129     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1130     \dim_gset:Nn \g_@@_dp_last_row_dim
1131     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1132   }
1133 }
1134 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1135 {
1136   \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.

```

1137 \str_if_empty:NTF \l_@@_name_str
1138 {
1139   \cs_if_exist:cT { @@_last_row_ \int_use:N \g_@@_env_int }
1140   {
1141     \int_set:Nn \l_@@_last_row_int
1142     { \use:c { @@_last_row_ \int_use:N \g_@@_env_int } }
1143   }
1144 }
1145 {
1146   \cs_if_exist:cT { @@_last_row_ \l_@@_name_str }
1147   {
1148     \int_set:Nn \l_@@_last_row_int
1149     { \use:c { @@_last_row_ \l_@@_name_str } }
1150   }

```

```

1151     }
1152 }

```

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).

```

1153 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1154 {
1155     \str_if_empty:NTF \l_@@_name_str
1156     {
1157         \cs_if_exist:cT { @@_last_col_ \int_use:N \g_@@_env_int }
1158         {
1159             \int_set:Nn \l_@@_last_col_int
1160             { \use:c { @@_last_col_ \int_use:N \g_@@_env_int } }
1161         }
1162     }
1163     {
1164         \cs_if_exist:cT { @@_last_col_ \l_@@_name_str }
1165         {
1166             \int_set:Nn \l_@@_last_col_int
1167             { \use:c { @@_last_col_ \l_@@_name_str } }
1168         }
1169     }
1170 }

```

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

```

1171 \@@_pre_array:

```

We compute the width of the two delimiters.

```

1172 \dim_zero_new:N \l_@@_left_delim_dim
1173 \dim_zero_new:N \l_@@_right_delim_dim
1174 \bool_if:NTF \l_@@_NiceArray_bool
1175 {
1176     \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1177     \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1178 }
1179 {

```

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

```

1180 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 #1 $ }
1181 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1182 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 #2 $ }
1183 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1184 }

```

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

```

1185 \box_clear_new:N \l_@@_the_array_box

```

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

```

1186 \bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:

```

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

```

1187 \@@_construct_preamble:n { #4 }

```

Now, the preamble is constructed in `\g_@@_preamble_tl`

Here is the beginning of the box which will contain the array. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the second part of the environment (and the closing `\c_math_toggle_token` also).

```

1188 \hbox_set:Nw \l_@@_the_array_box

```

```

1189 \skip_horizontal:N \l_@@_left_margin_dim
1190 \skip_horizontal:N \l_@@_extra_left_margin_dim
1191 \c_math_toggle_token
1192 \bool_if:NTF \l_@@_light_syntax_bool
1193 { \use:c { @@-light-syntax } }
1194 { \use:c { @@-normal-syntax } }
1195 }
1196 {
1197 \bool_if:NTF \l_@@_light_syntax_bool
1198 { \use:c { end @@-light-syntax } }
1199 { \use:c { end @@-normal-syntax } }
1200 \c_math_toggle_token
1201 \skip_horizontal:N \l_@@_right_margin_dim
1202 \skip_horizontal:N \l_@@_extra_right_margin_dim
1203 \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).

```

1204 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1205 {
1206 \bool_if:NF \l_@@_last_row_without_value_bool
1207 {
1208 \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1209 {
1210 \@@_error:n { Wrong~last~row }
1211 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1212 }
1213 }
1214 }

```

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”.³⁹

```

1215 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1216 \bool_if:NTF \g_@@_last_col_found_bool
1217 { \int_gdecr:N \c@jCol }
1218 {
1219 \int_compare:nNnT \l_@@_last_col_int > { -1 }
1220 { \@@_error:n { last~col~not~used } }
1221 }
1222 \bool_if:NF \l_@@_Matrix_bool
1223 {
1224 \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1225 { \@@_error:n { columns~not~used } }
1226 }

```

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

```

1227 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1228 \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. 90).

```

1229 \int_compare:nNnT \l_@@_first_col_int = 0
1230 {
1231 \skip_horizontal:N \arraycolsep
1232 \skip_horizontal:N \g_@@_width_first_col_dim
1233 }

```

³⁹We remind that the potential “first column” (exterior) has the number 0.

The construction of the real box is different when `\l_@@_NiceArray_bool` is true (`{NiceArray}` or `{NiceTabular}`) and in the other environments because, in `{NiceArray}` or `{NiceTabular}`, we have no delimiter to put. We begin with this case.

```

1234 \bool_if:NTF \l_@@_NiceArray_bool
1235 {
1236   \str_case:VnF \l_@@_baseline_str
1237   {
1238     b \@@_use_arraybox_with_notes_b:
1239     c \@@_use_arraybox_with_notes_c:
1240   }
1241   \@@_use_arraybox_with_notes:
1242 }

```

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).

```

1243 {
1244   \int_compare:nNnTF \l_@@_first_row_int = 0
1245   {
1246     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1247     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1248   }
1249   { \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”.⁴⁰

```

1250   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1251   {
1252     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1253     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1254   }
1255   { \dim_zero:N \l_tmpb_dim }
1256   \hbox_set:Nn \l_tmpa_box
1257   {
1258     \c_math_toggle_token
1259     \left #1
1260     \vcenter
1261     {

```

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.

```

1262     \skip_vertical:N -\l_tmpa_dim
1263     \skip_vertical:N -\arrayrulewidth
1264     \hbox
1265     {
1266       \bool_if:NTF \l_@@_NiceTabular_bool
1267       { \skip_horizontal:N -\tabcolsep }
1268       { \skip_horizontal:N -\arraycolsep }
1269       \@@_use_arraybox_with_notes_c:
1270       \bool_if:NTF \l_@@_NiceTabular_bool
1271       { \skip_horizontal:N -\tabcolsep }
1272       { \skip_horizontal:N -\arraycolsep }
1273     }

```

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

```

1274     \skip_vertical:N -\l_tmpb_dim
1275     \skip_vertical:N \arrayrulewidth
1276   }
1277   \right #2
1278   \c_math_toggle_token
1279 }

```

Now, the box `\l_tmpa_box` is created with the correct delimiters.

⁴⁰A 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).

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.

```

1280     \bool_if:NTF \l_@@_max_delimiter_width_bool
1281     { \@@_put_box_in_flow_bis:nn { #1 } { #2 } }
1282     \@@_put_box_in_flow:
1283 }

```

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. 90).

```

1284     \bool_if:NT \g_@@_last_col_found_bool
1285     {
1286         \skip_horizontal:N \g_@@_width_last_col_dim
1287         \skip_horizontal:N \arraycolsep
1288     }
1289     \@@_after_array:
1290     \egroup
1291     \bool_if:NT \c_@@_footnote_bool \endsavenotes
1292 }

```

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

We construct the preamble of the array

The transformation of the preamble is an operation in several steps.

The argument of `\@@_construct_preamble:n` is the preamble as given by the final user to the environment `{NiceTabular}` (or a variant). The preamble will be constructed in `\g_@@_preamble_tl`.

```

1293 \cs_new_protected:Npn \@@_construct_preamble:n #1
1294 {

```

First, we will do an “expansion” of the preamble with the tools of the package `array` itself. This “expansion” will expand all the constructions with `*` and with all column types (defined by the user or by various packages using `\newcolumntype`).

Since we use the tools of `array` to do this expansion, we will have a programmation which is not in the style of `expl3`.

We redefine the column types `w` and `W`. We use `\@@_newcolumntype` instead of `\newcolumntype` because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters `w` and `W`. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```

1295     \group_begin:

```

If we are in an environment without explicit preamble, we have nothing to do (excepted the treatment on both sides of the preamble which will be done at the end).

```

1296     \bool_if:NTF \l_@@_Matrix_bool
1297     { \tl_gset:Nn \g_@@_preamble_tl { #1 } }
1298     {
1299         \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1300         \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

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

```

1301         \@temptokena { #1 }

```

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

```

1302         \@tempswatrue

```

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

```

1303         \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

Now, we have to “patch” that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the “expansion”) following by a marker `\q_stop` and we will consume these tokens constructing the (new form of the) preamble in `\g_@@_preamble_tl`. This is done recursively with the command `\@@_patch_preamble:n`. In the same time, we will count the columns with the counter `\c@jCol`.

```

1304     \int_gzero_new:N \c@jCol
1305     \bool_if:NTF \l_@@_vlines_bool
1306     {
1307         \tl_gset:Nn \g_@@_preamble_tl
1308         { ! { \skip_horizontal:N \arrayrulewidth } }
1309     }
1310     { \tl_gclear:N \g_@@_preamble_tl }

```

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

```

1311     \int_zero:N \l_tmpa_int

```

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

```

1312     \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1313     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1314 }

```

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

```

1315     \bool_if:NT \l_@@_colortbl_like_bool
1316     {
1317         \regex_replace_all:NnN
1318         \c_@@_columncolor_regex
1319         { \c { @@_columncolor_preamble } }
1320         \g_@@_preamble_tl
1321     }

```

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

```

1322     \int_compare:nNnTF \l_@@_first_col_int = 0
1323     { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1324     {
1325         \bool_lazy_all:nT
1326         {
1327             \l_@@_NiceArray_bool
1328             { \bool_not_p:n \l_@@_NiceTabular_bool }
1329             { \bool_not_p:n \l_@@_vlines_bool }
1330             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1331         }
1332         { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1333     }
1334     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1335     { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1336     {
1337         \bool_lazy_all:nT
1338         {
1339             \l_@@_NiceArray_bool
1340             { \bool_not_p:n \l_@@_NiceTabular_bool }
1341             { \bool_not_p:n \l_@@_vlines_bool }
1342             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1343         }
1344         { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1345     }

```

We add a last column to raise a good error message when the user put more columns than allowed by its preamble. However, for technical reasons, it’s not possible to do that in `{NiceTabular*}` (`\l_@@_tabular_width_dim=0pt`).

```

1346     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1347     {
1348         \tl_gput_right:Nn
1349         \g_@@_preamble_tl

```

```

1350         { > { \@@_error_too_much_cols: } 1 }
1351     }

```

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

```

1352     \group_end:
1353 }

```

```

1354 \cs_new_protected:Npn \@@_patch_preamble:n #1
1355 {
1356     \str_case:nnF { #1 }
1357     {
1358         c { \@@_patch_preamble_i:n #1 }
1359         l { \@@_patch_preamble_i:n #1 }
1360         r { \@@_patch_preamble_i:n #1 }
1361         > { \@@_patch_preamble_ii:nn #1 }
1362         ! { \@@_patch_preamble_ii:nn #1 }
1363         @ { \@@_patch_preamble_ii:nn #1 }
1364         | { \@@_patch_preamble_iii:n #1 }
1365         p { \@@_patch_preamble_iv:nnn t #1 }
1366         m { \@@_patch_preamble_iv:nnn c #1 }
1367         b { \@@_patch_preamble_iv:nnn b #1 }
1368         \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
1369         \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1370         \@@_true_c: { \@@_patch_preamble_vi:n #1 }
1371         \q_stop { }
1372     }
1373     {
1374         \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1375         { \@@_patch_preamble_vii:n #1 }
1376         { \@@_fatal:nn { unknown~column~type } { #1 } }
1377     }
1378 }

```

For c, l and r

```

1379 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1380 {
1381     \tl_gput_right:Nn \g_@@_preamble_tl { > \@@_Cell: #1 < \@@_end_Cell: }

```

We increment the counter of columns.

```

1382     \int_gincr:N \c@jCol
1383     \@@_patch_preamble_viii:n
1384 }

```

For >, ! and @

```

1385 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1386 {
1387     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1388     \@@_patch_preamble:n
1389 }

```

For |

```

1390 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1391 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1392     \int_incr:N \l_tmpa_int
1393     \@@_patch_preamble_iii_i:n
1394 }

```

```

1395 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1396 {
1397   \str_if_eq:nnTF { #1 } |
1398   { \@@_patch_preamble_iii:n | }
1399   {
1400     \tl_gput_right:Nx \g_@@_preamble_tl
1401     {
1402       \exp_not:N !
1403       {
1404         \skip_horizontal:n
1405         {
1406           \dim_eval:n
1407           {
1408             \arrayrulewidth * \l_tmpa_int
1409             + \doublerulesep * ( \l_tmpa_int - 1)
1410           }
1411         }
1412       }
1413     }
1414     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1415     { \@@_vline:nn { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } }
1416     \int_zero:N \l_tmpa_int
1417     \@@_patch_preamble:n #1
1418   }
1419 }

```

For p, m and b

```

1420 \cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
1421 {
1422   \tl_gput_right:Nn \g_@@_preamble_tl
1423   {
1424     > {
1425       \@@_Cell:
1426       \begin { minipage } [ #1 ] { #3 }
1427       \mode_leave_vertical:
1428       \box_use:N \@arstrutbox
1429     }
1430     c
1431     < { \box_use:N \@arstrutbox \end { minipage } \@@_end_Cell: }
1432   }

```

We increment the counter of columns.

```

1433   \int_gincr:N \c@jCol
1434   \@@_patch_preamble_viii:n
1435 }

```

For w and W

```

1436 \cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
1437 {
1438   \tl_gput_right:Nn \g_@@_preamble_tl
1439   {
1440     > {
1441       \hbox_set:Nw \l_@@_cell_box
1442       \@@_Cell:
1443     }
1444     c
1445     < {
1446       \@@_end_Cell:
1447       #1
1448       \hbox_set_end:
1449       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
1450     }
1451   }

```


We increment the counter of columns.

```

1452 \int_gincr:N \c@jCol
1453 \@@_patch_preamble_viii:n
1454 }

```

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

```

1455 \cs_new_protected:Npn \@@_patch_preamble_vi:n #1
1456 {
1457 \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

We increment the counter of columns.

```

1458 \int_gincr:N \c@jCol
1459 \@@_patch_preamble_viii:n
1460 }

1461 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
1462 {
1463 \tl_gput_right:Nn \g_@@_preamble_tl
1464 { ! { \skip_horizontal:N 2\l_@@_radius_dim } }

```

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

```

1465 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1466 { \@@_vdottedline:n { \int_use:N \c@jCol } }
1467 \@@_patch_preamble:n
1468 }

```

After a specifier of column, we have to test whether there is one or several `<{. .}` because, after those potential `<{. .}`, we have to insert `!\skip_horizontal:N ...` when the key `vlines` is used.

```

1469 \cs_new_protected:Npn \@@_patch_preamble_viii:n #1
1470 {
1471 \str_if_eq:nnTF { #1 } { < }
1472 \@@_patch_preamble_ix:n
1473 {
1474 \bool_if:NT \l_@@_vlines_bool
1475 {
1476 \tl_gput_right:Nn \g_@@_preamble_tl
1477 { ! { \skip_horizontal:N \arrayrulewidth } }
1478 }
1479 \@@_patch_preamble:n { #1 }
1480 }
1481 }

1482 \cs_new_protected:Npn \@@_patch_preamble_ix:n #1
1483 {
1484 \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
1485 \@@_patch_preamble_viii:n
1486 }

```

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`).

```

1487 \cs_new_protected:Npn \@@_put_box_in_flow:
1488 {
1489 \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
1490 \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
1491 \str_if_eq:VnTF \l_@@_baseline_str { c }
1492 { \box_use_drop:N \l_tmpa_box }
1493 \@@_put_box_in_flow_i:
1494 }

```

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).

```

1495 \cs_new_protected:Npn \@@_put_box_in_flow_i:
1496 {
1497   \pgfpicture
1498     \@@_qpoint:n { row - 1 }
1499     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1500     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
1501     \dim_gadd:Nn \g_tmpa_dim \pgf@y
1502     \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).

```

1503   \str_if_in:NnTF \l_@@_baseline_str { line- }
1504   {
1505     \int_set:Nn \l_tmpa_int
1506     {
1507       \str_range:Nnn
1508         \l_@@_baseline_str
1509         6
1510         { \str_count:N \l_@@_baseline_str }
1511     }
1512     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
1513   }
1514   {
1515     \str_case:VnF \l_@@_baseline_str
1516     {
1517       { t } { \int_set:Nn \l_tmpa_int 1 }
1518       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
1519     }
1520     { \int_set:Nn \l_tmpa_int \l_@@_baseline_str }
1521     \bool_lazy_or:nnTF
1522     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
1523     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
1524     {
1525       \@@_error:n { bad-value-for-baseline }
1526       \int_set:Nn \l_tmpa_int 1
1527     }
1528     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

1529     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
1530   }
1531   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

1532   \endpgfpicture
1533   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
1534   \box_use_drop:N \l_tmpa_box
1535 }

```

```

1536 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
1537 {
1538   \int_compare:nNnTF \c@tabularnote = 0
1539   { \box_use_drop:N \l_@@_the_array_box }
1540   {
1541     \begin { minipage } { \box_wd:N \l_@@_the_array_box }
1542     \box_use_drop:N \l_@@_the_array_box
1543     \skip_vertical:N 0.65ex

```

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

```

1544   \group_begin:
1545   \l_@@_notes_code_before_tl

```

We compose the tabular notes with a list of `enumitem`. The `\strut` and the `\unskip` are designed to give the ability to put a `\bottomrule` at the end of the notes with a good vertical space.

```

1546     \bool_if:NTF \l_@@_notes_para_bool
1547     {
1548         \begin { tabularnotes* }
1549         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
1550         \end { tabularnotes* }

```

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

```

1551     \par
1552 }
1553 {
1554     \tabularnotes
1555     \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
1556     \endtabularnotes
1557 }
1558 \unskip
1559 \group_end:
1560 \bool_if:NT \l_@@_notes_bottomrule_bool
1561 {
1562     \bool_if:NTF \c_@@_booktabs_loaded_bool
1563     {

```

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

```

1564     \skip_vertical:N \aboverulesep

```

`\CT@arc@` is the specification of color defined by `colortbl` but you use it even if `colortbl` is not loaded.

```

1565         { \CT@arc@ \hrule height \heavyrulewidth }
1566     }
1567     { \@@_error:n { bottomule~without~booktabs } }
1568 }
1569 \l_@@_notes_code_after_tl
1570 \end { minipage }
1571 \seq_gclear:N \g_@@_tabularnotes_seq
1572 \int_gzero:N \c@tabularnote
1573 }
1574 }

```

The case of `baseline` equal to `b`. Remember that, when the key `b` is used, the `{array}` (of array) is constructed with the option `t` (and not `b`). Now, we do the translation to take into account the option `b`.

```

1575 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
1576 {
1577     \pgfpicture
1578     \@@_qpoint:n { row - 1 }
1579     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1580     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
1581     \dim_gsub:Nn \g_tmpa_dim \pgf@y
1582     \endpgfpicture
1583     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
1584     \int_compare:nNnT \l_@@_first_row_int = 0
1585     {
1586         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
1587         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
1588     }
1589     \box_move_up:nn \g_tmpa_dim { \@@_use_arraybox_with_notes_c: }
1590 }

```

Now, the general case (hence the `g` in the name).

```

1591 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
1592 {

```

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

```
1593 \str_if_eq:VnT \l_@@_baseline_str { t }
1594 { \tl_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`.

```
1595 \int_set:Nn \l_tmpa_int \l_@@_baseline_str
1596 \bool_lazy_or:nnT
1597 { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
1598 { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
1599 {
1600   \@@_error:n { bad~value~for~baseline }
1601   \int_set:Nn \l_tmpa_int 1
1602 }
1603 \pgfpicture
1604 \@@_qpoint:n { row - 1 }
1605 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1606 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
1607 \dim_gsub:Nn \g_tmpa_dim \pgf@y
1608 \endpgfpicture
1609 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
1610 \int_compare:nNnT \l_@@_first_row_int = 0
1611 {
1612   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
1613   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
1614 }
1615 \box_move_up:nn \g_tmpa_dim { \@@_use_arraybox_with_notes_c: }
1616 }
```

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.

```
1617 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
1618 {
```

We will compute the real width of both delimiters used.

```
1619 \dim_zero_new:N \l_@@_real_left_delim_dim
1620 \dim_zero_new:N \l_@@_real_right_delim_dim
1621 \hbox_set:Nn \l_tmpb_box
1622 {
1623   \c_math_toggle_token
1624   \left #1
1625   \vcenter
1626   {
1627     \vbox_to_ht:nn
1628     { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
1629     { }
1630   }
1631   \right .
1632   \c_math_toggle_token
1633 }
1634 \dim_set:Nn \l_@@_real_left_delim_dim
1635 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
1636 \hbox_set:Nn \l_tmpb_box
1637 {
1638   \c_math_toggle_token
1639   \left .
1640   \vbox_to_ht:nn
1641   { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
1642   { }
1643   \right #2
1644   \c_math_toggle_token
1645 }
```

```

1646 \dim_set:Nn \l_@@_real_right_delim_dim
1647 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }

```

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

```

1648 \skip_horizontal:N \l_@@_left_delim_dim
1649 \skip_horizontal:N -\l_@@_real_left_delim_dim
1650 \@@_put_box_in_flow:
1651 \skip_horizontal:N \l_@@_right_delim_dim
1652 \skip_horizontal:N -\l_@@_real_right_delim_dim
1653 }

```

The construction of the array in the environment `{NiceArrayWithDelims}` is, in fact, done by the environment `{@@-light-syntax}` or by the environment `{@@-normal-syntax}` (whether the option `light-syntax` is in force or not). When the key `light-syntax` is not used, the construction is a standard environment (and, thus, it's possible to use verbatim in the array).

```

1654 \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).

```

1655 {
1656   \peek_meaning_ignore_spaces:NTF \end \@@_analyze_end:Nn

```

Here is the call to `\array` (we have a dedicated macro `\@@_array`: because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

1657   { \exp_args:NV \@@_array: \g_@@_preamble_tl }
1658 }
1659 {
1660   \@@_create_col_nodes:
1661   \endarray
1662 }

```

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

```

1663 \NewDocumentEnvironment { @@-light-syntax } { b }
1664 {

```

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`.

```

1665   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
1666   \tl_map_inline:nn { #1 }
1667   {
1668     \tl_if_eq:nnT { ##1 } { & }
1669     { \@@_fatal:n { ampersand-in-light-syntax } }
1670     \tl_if_eq:nnT { ##1 } { \ }
1671     { \@@_fatal:n { double-backslash-in-light-syntax } }
1672   }

```

Now, you extract the `code-after` of the body of the environment. Maybe, there is no command `\CodeAfter` in the body. That's why you put a marker `\CodeAfter` after `#1`. If there is yet a `\CodeAfter` in `#1`, this second (or third...) `\CodeAfter` will be caught in the value of `\g_nicematrix_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to *no-op* before the execution of `\g_nicematrix_code_after_tl`.

```

1673   \@@_light_syntax_i #1 \CodeAfter \q_stop
1674 }

```

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`.

```

1675 { }

```

```

1676 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
1677 {
1678   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }

```

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

```

1679   \seq_gclear_new:N \g_@@_rows_seq
1680   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
1681   \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.

```

1682   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1683     { \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`).

```

1684   \exp_args:NV \@@_array: \g_@@_preamble_tl

```

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

```

1685   \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
1686   \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
1687   \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
1688   \@@_create_col_nodes:
1689   \endarray
1690 }

1691 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
1692 { \tl_if_empty:nF { #1 } { \ \ \ \@@_line_with_light_syntax_i:n { #1 } } }

1693 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
1694 {
1695   \seq_gclear_new:N \g_@@_cells_seq
1696   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
1697   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
1698   \l_tmpa_tl
1699   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
1700 }

```

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).

```

1701 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
1702 {
1703   \str_if_eq:VnT \g_@@_name_env_str { #2 }
1704     { \@@_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.

```

1705   \end { #2 }
1706 }

```

The command `\@@_create_col_nodes`: will construct a special last row. That last row is a false row used to create the `col` nodes and to fix the width of the columns (when the array is constructed with an option which specify the width of the columns).

```

1707 \cs_new:Npn \@@_create_col_nodes:
1708 {
1709   \crrc
1710   \int_compare:nNnT \l_@@_first_col_int = 0
1711     {
1712       \omit
1713       \skip_horizontal:N -2\col@sep
1714       \bool_if:NT \l_@@_code_before_bool
1715         { \pgfsys@markposition { \@@_env: - col - 0 } }

```

```

1716 \pgfpicture
1717 \pgfrememberpicturerepositiononpagetrue
1718 \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
1719 \str_if_empty:NF \l_@@_name_str
1720 { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
1721 \endpgfpicture
1722 &
1723 }
1724 \omit

```

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

```

1725 \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`).

```

1726 \int_compare:nNnTF \l_@@_first_col_int = 0
1727 {
1728   \bool_if:NT \l_@@_code_before_bool
1729   {
1730     \hbox
1731     {
1732       \skip_horizontal:N -0.5\arrayrulewidth
1733       \pgfsys@markposition { \@@_env: - col - 1 }
1734       \skip_horizontal:N 0.5\arrayrulewidth
1735     }
1736   }
1737   \pgfpicture
1738   \pgfrememberpicturerepositiononpagetrue
1739   \pgfcoordinate { \@@_env: - col - 1 }
1740   { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
1741   \str_if_empty:NF \l_@@_name_str
1742   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
1743   \endpgfpicture
1744 }
1745 {
1746   \bool_if:NT \l_@@_code_before_bool
1747   {
1748     \hbox
1749     {
1750       \skip_horizontal:N 0.5 \arrayrulewidth
1751       \pgfsys@markposition { \@@_env: - col - 1 }
1752       \skip_horizontal:N -0.5\arrayrulewidth
1753     }
1754   }
1755   \pgfpicture
1756   \pgfrememberpicturerepositiononpagetrue
1757   \pgfcoordinate { \@@_env: - col - 1 }
1758   { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
1759   \str_if_empty:NF \l_@@_name_str
1760   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
1761   \endpgfpicture
1762 }

```

We compute in `\g_tmpa_skip` the common width of the columns (it's a skip and not a dimension). We use a global variable because we are in a cell of an `\halign` and because we have to use this variable in other cells (of the same row). The affectation of `\g_tmpa_skip`, like all the affectations, must be done after the `\omit` of the cell.

We give a default value for `\g_tmpa_skip` (0 pt plus 1 fill) but it will just after be erased by a fixed value in the concerned cases.

```

1763 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
1764 \bool_if:NF \l_@@_auto_columns_width_bool
1765 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
1766 {
1767   \bool_lazy_and:nnTF

```

```

1768     \l_@@_auto_columns_width_bool
1769     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
1770     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
1771     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
1772     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
1773   }
1774   \skip_horizontal:N \g_tmpa_skip
1775   \hbox
1776   {
1777     \bool_if:NT \l_@@_code_before_bool
1778     {
1779       \hbox
1780       {
1781         \skip_horizontal:N -0.5\arrayrulewidth
1782         \pgfsys@markposition { \@@_env: - col - 2 }
1783         \skip_horizontal:N 0.5\arrayrulewidth
1784       }
1785     }
1786     \pgfpicture
1787     \pgfrememberpicturepositiononpagetrue
1788     \pgfcoordinate { \@@_env: - col - 2 }
1789     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
1790     \str_if_empty:NF \l_@@_name_str
1791     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
1792     \endpgfpicture
1793   }

```

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.

```

1794   \int_gset:Nn \g_tmpa_int 1
1795   \bool_if:NTF \g_@@_last_col_found_bool
1796   { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
1797   { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
1798   {
1799     &
1800     \omit

```

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

```

1801     \int_gincr:N \g_tmpa_int
1802     \skip_horizontal:N \g_tmpa_skip
1803     \bool_if:NT \l_@@_code_before_bool
1804     {
1805       \hbox
1806       {
1807         \skip_horizontal:N -0.5\arrayrulewidth
1808         \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1809         \skip_horizontal:N 0.5\arrayrulewidth
1810       }
1811     }

```

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

```

1812     \pgfpicture
1813     \pgfrememberpicturepositiononpagetrue
1814     \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1815     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
1816     \str_if_empty:NF \l_@@_name_str
1817     {
1818       \pgfnodealias
1819       { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
1820       { \@@_env: - col - \@@_succ:n \g_tmpa_int }
1821     }
1822     \endpgfpicture
1823   }
1824   \bool_if:NT \g_@@_last_col_found_bool
1825   {

```



```

1826 \bool_if:NT \l_@@_code_before_bool
1827 {
1828   \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1829 }
1830 \skip_horizontal:N 2\col@sep
1831 \pgfpicture
1832 \pgfrememberpicturepositiononpagetrue
1833 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1834 \pgfpointorigin
1835 \str_if_empty:NF \l_@@_name_str
1836 {
1837   \pgfnodealias
1838   { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
1839   { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
1840 }
1841 \endpgfpicture
1842 \skip_horizontal:N -2\col@sep
1843 }
1844 \cr
1845 }

```

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

```

1846 \tl_const:Nn \c_@@_preamble_first_col_tl
1847 {
1848   >
1849   {
1850     \@@_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.

```

1851 \hbox_set:Nw \l_@@_cell_box
1852 \@@_math_toggle_token:
1853 \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”.

```

1854 \bool_lazy_and:nnT
1855 { \int_compare_p:nNn \c@iRow > 0 }
1856 {
1857   \bool_lazy_or_p:nn
1858   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
1859   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
1860 }
1861 {
1862   \l_@@_code_for_first_col_tl
1863   \xglobal \colorlet { nicematrix-first-col } { . }
1864 }
1865 }

```

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`.

```

1866 l
1867 <
1868 {
1869   \@@_math_toggle_token:
1870   \hbox_set_end:
1871   \@@_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.

```

1872 \dim_gset:Nn \g_@@_width_first_col_dim
1873 { \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.

```

1874 \hbox_overlap_left:n
1875 {
1876   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1877     \@@_node_for_the_cell:
1878     { \box_use_drop:N \l_@@_cell_box }
1879     \skip_horizontal:N \l_@@_left_delim_dim
1880     \skip_horizontal:N \l_@@_left_margin_dim
1881     \skip_horizontal:N \l_@@_extra_left_margin_dim
1882   }
1883   \skip_horizontal:N -2\col@sep
1884 }
1885 }

```

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

```

1886 \tl_const:Nn \c_@@_preamble_last_col_tl
1887 {
1888   >
1889   {

```

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

```

1890 \bool_gset_true:N \g_@@_last_col_found_bool
1891 \int_gincr:N \c@jCol
1892 \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.

```

1893 \hbox_set:Nw \l_@@_cell_box
1894 \@@_math_toggle_token:
1895 \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”.

```

1896 \int_compare:nNnT \c@iRow > 0
1897 {
1898   \bool_lazy_or:nnT
1899   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
1900   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
1901   {
1902     \l_@@_code_for_last_col_tl
1903     \xglobal \colorlet { nicematrix-last-col } { . }
1904   }
1905 }
1906 }
1907 1
1908 <
1909 {
1910 \@@_math_toggle_token:
1911 \hbox_set_end:
1912 \@@_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.

```

1913 \dim_gset:Nn \g_@@_width_last_col_dim
1914 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
1915 \skip_horizontal:N -2\col@sep

```

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

```

1916 \hbox_overlap_right:n
1917 {
1918   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1919   {
1920     \skip_horizontal:N \l_@@_right_delim_dim
1921     \skip_horizontal:N \l_@@_right_margin_dim
1922     \skip_horizontal:N \l_@@_extra_right_margin_dim

```

```

1923         \@@_node_for_the_cell:
1924     }
1925 }
1926 }
1927 }

```

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.

```

1928 \NewDocumentEnvironment { NiceArray } { }
1929 {
1930     \bool_set_true:N \l_@@_NiceArray_bool
1931     \str_if_empty:NT \g_@@_name_env_str
1932     { \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).

```

1933     \NiceArrayWithDelims . .
1934 }
1935 { \endNiceArrayWithDelims }

```

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

```

1936 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
1937 {
1938     \NewDocumentEnvironment { #1 NiceArray } { }
1939     {
1940         \str_if_empty:NT \g_@@_name_env_str
1941         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
1942         \@@_test_if_math_mode:
1943         \NiceArrayWithDelims #2 #3
1944     }
1945     { \endNiceArrayWithDelims }
1946 }
1947 \@@_def_env:nnn p ( )
1948 \@@_def_env:nnn b [ ]
1949 \@@_def_env:nnn B \{ \}
1950 \@@_def_env:nnn v | |
1951 \@@_def_env:nnn V \l \r

```

The environment `{NiceMatrix}` and its variants

```

1952 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
1953 {
1954     \bool_set_true:N \l_@@_Matrix_bool
1955     \use:c { #1 NiceArray }
1956     {
1957         *
1958         {
1959             \int_compare:nNnTF \l_@@_last_col_int < 0
1960             \c@MaxMatrixCols
1961             { \@@_pred:n \l_@@_last_col_int }
1962         }
1963         { > \@@_Cell: #2 < \@@_end_Cell: }
1964     }
1965 }
1966 \clist_map_inline:nn { { } , p , b , B , v , V }
1967 {
1968     \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }

```

```

1969 {
1970   \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
1971   \tl_set:Nn \l_@@_type_of_col_tl c
1972   \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
1973   \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
1974 }
1975 { \use:c { end #1 NiceArray } }
1976 }

```

The environments `{NiceTabular}` and `{NiceTabular*}`

```

1977 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
1978 {
1979   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
1980   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
1981   \bool_set_true:N \l_@@_NiceTabular_bool
1982   \NiceArray { #2 }
1983 }
1984 { \endNiceArray }

1985 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
1986 {
1987   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
1988   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
1989   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
1990   \bool_set_true:N \l_@@_NiceTabular_bool
1991   \NiceArray { #3 }
1992 }
1993 { \endNiceArray }

```

After the construction of the array

```

1994 \cs_new_protected:Npn \@@_after_array:
1995 {
1996   \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.

```

1997   \bool_if:NT \g_@@_last_col_found_bool
1998   { \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`.

```

1999   \bool_if:NT \l_@@_last_col_without_value_bool
2000   {
2001     \dim_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int
2002     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2003     \iow_shipout:Nx \@mainaux
2004     {
2005       \cs_gset:cpn { @@_last_col_ \int_use:N \g_@@_env_int }
2006       { \int_use:N \g_@@_col_total_int }
2007     }
2008     \str_if_empty:NF \l_@@_name_str
2009     {
2010       \iow_shipout:Nx \@mainaux
2011       {
2012         \cs_gset:cpn { @@_last_col_ \l_@@_name_str }
2013         { \int_use:N \g_@@_col_total_int }

```

```

2014     }
2015   }
2016   \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2017 }

```

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.

```

2018   \bool_if:NT \l_@@_last_row_without_value_bool
2019   {
2020     \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.

```

2021     \bool_if:NF \l_@@_light_syntax_bool
2022     {
2023       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2024       \iow_shipout:Nx \@mainaux
2025       {
2026         \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int }
2027         { \int_use:N \g_@@_row_total_int }
2028       }

```

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.

```

2029       \str_if_empty:NF \l_@@_name_str
2030       {
2031         \iow_shipout:Nx \@mainaux
2032         {
2033           \cs_gset:cpn { @@_last_row_ \l_@@_name_str }
2034           { \int_use:N \g_@@_row_total_int }
2035         }
2036       }
2037       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2038     }
2039   }

```

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

```

2040   \bool_if:NT \l_@@_code_before_bool
2041   {
2042     \iow_now:Nn \@mainaux \ExplSyntaxOn
2043     \iow_now:Nx \@mainaux
2044     { \seq_clear_new:c { @@_size _ \int_use:N \g_@@_env_int _ seq } }
2045     \iow_now:Nx \@mainaux
2046     {
2047       \seq_gset_from_clist:cn { @@_size _ \int_use:N \g_@@_env_int _ seq }
2048       {
2049         \int_use:N \l_@@_first_row_int ,
2050         \int_use:N \g_@@_row_total_int ,
2051         \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`.

```

2052       \bool_lazy_and:nnTF
2053       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
2054       { \bool_not_p:n \g_@@_last_col_found_bool }
2055       \@@_succ:n
2056       \int_use:N
2057       \g_@@_col_total_int
2058     }

```

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

```

2059     \seq_gset_from_clist:cn
2060     { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
2061     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
2062   }

```

```

2063     \iow_now:Nn \@mainaux \ExplSyntaxOff
2064 }

```

By default, the diagonal lines will be parallelized⁴¹. 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.

```

2065     \bool_if:NT \l_@@_parallelize_diags_bool
2066     {
2067         \int_gzero_new:N \g_@@_ddots_int
2068         \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.

```

2069         \dim_gzero_new:N \g_@@_delta_x_one_dim
2070         \dim_gzero_new:N \g_@@_delta_y_one_dim
2071         \dim_gzero_new:N \g_@@_delta_x_two_dim
2072         \dim_gzero_new:N \g_@@_delta_y_two_dim
2073     }
2074     \bool_if:nTF \l_@@_medium_nodes_bool
2075     {
2076         \bool_if:nTF \l_@@_large_nodes_bool
2077         \@@_create_medium_and_large_nodes:
2078         \@@_create_medium_nodes:
2079     }
2080     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
2081     \int_zero_new:N \l_@@_initial_i_int
2082     \int_zero_new:N \l_@@_initial_j_int
2083     \int_zero_new:N \l_@@_final_i_int
2084     \int_zero_new:N \l_@@_final_j_int
2085     \bool_set_false:N \l_@@_initial_open_bool
2086     \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.

```

2087     \bool_if:NT \l_@@_small_bool
2088     {
2089         \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
2090         \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.

```

2091         \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
2092     }

```

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

```

2093     \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it will do nothing.

```

2094     \@@_compute_corners:

```

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

```

2095     % \bool_lazy_all:nT
2096     % {
2097     %     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }

```

⁴¹It's possible to use the option `parallelize-diags` to disable this parallelization.

```

2098 % { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
2099 % { \seq_if_empty_p:N \l_@@_empty_corner_cells_seq }
2100 % }
2101 % {
2102 % \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn
2103 % \cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn
2104 % }
2105 \bool_if:NT \l_@@_hlines_bool \@@_draw_hlines:
2106 \bool_if:NT \l_@@_vlines_bool \@@_draw_vlines:
2107 \g_@@_internal_code_after_tl
2108 \tl_gclear:N \g_@@_internal_code_after_tl

```

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

```

2109 \cs_set_eq:NN \ialign \@@_old_ialign:
2110 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:

```

Now, the code-after.

```

2111 \bool_if:NT \c_@@_tikz_loaded_bool
2112 {
2113   \tikzset
2114   {
2115     every~picture / .style =
2116     {
2117       overlay ,
2118       remember~picture ,
2119       name~prefix = \@@_env: -
2120     }
2121   }
2122 }
2123 \cs_set_eq:NN \line \@@_line

```

When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it's possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That's why we set `\Code-after` to be *no-op* now.

```

2124 \cs_set_eq:NN \CodeAfter \prg_do_nothing:

```

And here's the code-after:

```

2125 \g_nicematrix_code_after_tl
2126 \tl_gclear:N \g_nicematrix_code_after_tl
2127 \group_end:

```

`\g_nicematrix_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor` (when the key `colortbl-like` is in force). These instructions will be written on the `aux` file to be added to the code-before in the next run.

```

2128 \tl_if_empty:NF \g_nicematrix_code_before_tl
2129 {

```

The command `\rowcolor` in tabular will in fact use `\rectanglecolor` in order to follow the behaviour of `\rowcolor` of `colortbl`. That's why there may be a command `\rectanglecolor` in `\g_nicematrix_code_before_tl`. In order to avoid an error during the expansion, we define a protected version of `\rectanglecolor`.

```

2130 \cs_set_protected:Npn \rectanglecolor { }
2131 \cs_set_protected:Npn \columncolor { }
2132 \iow_now:Nn \@mainaux \ExplSyntaxOn
2133 \iow_now:Nx \@mainaux
2134 {
2135   \tl_gset:cn
2136   { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
2137   { \g_nicematrix_code_before_tl }
2138 }
2139 \iow_now:Nn \@mainaux \ExplSyntaxOff
2140 \bool_set_true:N \l_@@_code_before_bool
2141 }

```

```

2142 \str_gclear:N \g_@@_name_env_str
2143 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁴². This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```

2144 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
2145 }

```

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).

```

2146 \AtBeginDocument
2147 {
2148   \cs_new_protected:Npx \@@_draw_dotted_lines:
2149   {
2150     \c_@@_pgfortikzpicture_tl
2151     \@@_draw_dotted_lines_i:
2152     \c_@@_endpgfortikzpicture_tl
2153   }
2154 }

```

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

```

2155 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
2156 {
2157   \pgfrememberpicturepositiononpagetrue
2158   \pgf@relevantforpicturesizefalse
2159   \g_@@_HVdotsfor_lines_tl
2160   \g_@@_Vdots_lines_tl
2161   \g_@@_Ddots_lines_tl
2162   \g_@@_Iddots_lines_tl
2163   \g_@@_Cdots_lines_tl
2164   \g_@@_Ldots_lines_tl
2165 }

2166 \cs_new_protected:Npn \@@_restore_iRow_jCol:
2167 {
2168   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
2169   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
2170 }

```

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.

⁴²e.g. `\color[rgb]{0.5,0.5,0}`)

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.

```
2171 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
2172 {
```

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

```
2173 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
2174 \int_set:Nn \l_@@_initial_i_int { #1 }
2175 \int_set:Nn \l_@@_initial_j_int { #2 }
2176 \int_set:Nn \l_@@_final_i_int { #1 }
2177 \int_set:Nn \l_@@_final_j_int { #2 }
```

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

```
2178 \bool_set_false:N \l_@@_stop_loop_bool
2179 \bool_do_until:Nn \l_@@_stop_loop_bool
2180 {
2181   \int_add:Nn \l_@@_final_i_int { #3 }
2182   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
2183 \bool_set_false:N \l_@@_final_open_bool
2184 \int_compare:nNnTF \l_@@_final_i_int > \c@iRow
2185 {
2186   \int_compare:nNnTF { #3 } = 1
2187   { \bool_set_true:N \l_@@_final_open_bool }
2188   {
2189     \int_compare:nNnT \l_@@_final_j_int > \c@jCol
2190     { \bool_set_true:N \l_@@_final_open_bool }
2191   }
2192 }
2193 {
2194   \int_compare:nNnTF \l_@@_final_j_int < 1
2195   {
2196     \int_compare:nNnT { #4 } = { -1 }
2197     { \bool_set_true:N \l_@@_final_open_bool }
2198   }
2199   {
2200     \int_compare:nNnT \l_@@_final_j_int > \c@jCol
2201     {
2202       \int_compare:nNnT { #4 } = 1
2203       { \bool_set_true:N \l_@@_final_open_bool }
2204     }
2205   }
2206 }
2207 \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.

```
2208 {
```

We do a step backwards.

```
2209 \int_sub:Nn \l_@@_final_i_int { #3 }
2210 \int_sub:Nn \l_@@_final_j_int { #4 }
2211 \bool_set_true:N \l_@@_stop_loop_bool
2212 }
```

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`.

```

2213     {
2214         \cs_if_exist:cTF
2215         {
2216             @@ _ dotted _
2217             \int_use:N \l_@@_final_i_int -
2218             \int_use:N \l_@@_final_j_int
2219         }
2220         {
2221             \int_sub:Nn \l_@@_final_i_int { #3 }
2222             \int_sub:Nn \l_@@_final_j_int { #4 }
2223             \bool_set_true:N \l_@@_final_open_bool
2224             \bool_set_true:N \l_@@_stop_loop_bool
2225         }
2226         {
2227             \cs_if_exist:cTF
2228             {
2229                 pgf @ sh @ ns @ \@@_env:
2230                 - \int_use:N \l_@@_final_i_int
2231                 - \int_use:N \l_@@_final_j_int
2232             }
2233             { \bool_set_true:N \l_@@_stop_loop_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the 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.

```

2234         {
2235             \cs_set:cpn
2236             {
2237                 @@ _ dotted _
2238                 \int_use:N \l_@@_final_i_int -
2239                 \int_use:N \l_@@_final_j_int
2240             }
2241             { }
2242         }
2243     }
2244 }
2245

```

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

```

2246 \bool_set_false:N \l_@@_stop_loop_bool
2247 \bool_do_until:Nn \l_@@_stop_loop_bool
2248 {
2249     \int_sub:Nn \l_@@_initial_i_int { #3 }
2250     \int_sub:Nn \l_@@_initial_j_int { #4 }
2251     \bool_set_false:N \l_@@_initial_open_bool
2252     \int_compare:nNnTF \l_@@_initial_i_int < 1
2253     {
2254         \int_compare:nNnTF { #3 } = 1
2255         { \bool_set_true:N \l_@@_initial_open_bool }
2256         {
2257             \int_compare:nNnT \l_@@_initial_j_int = 0
2258             { \bool_set_true:N \l_@@_initial_open_bool }
2259         }
2260     }
2261     {
2262         \int_compare:nNnTF \l_@@_initial_j_int < 1

```

```

2263     {
2264         \int_compare:nNtT { #4 } = 1
2265         { \bool_set_true:N \l_@@_initial_open_bool }
2266     }
2267     {
2268         \int_compare:nNtT \l_@@_initial_j_int > \c@jCol
2269         {
2270             \int_compare:nNtT { #4 } = { -1 }
2271             { \bool_set_true:N \l_@@_initial_open_bool }
2272         }
2273     }
2274 }
2275 \bool_if:NTF \l_@@_initial_open_bool
2276 {
2277     \int_add:Nn \l_@@_initial_i_int { #3 }
2278     \int_add:Nn \l_@@_initial_j_int { #4 }
2279     \bool_set_true:N \l_@@_stop_loop_bool
2280 }
2281 {
2282     \cs_if_exist:cTF
2283     {
2284         @@ _ dotted _
2285         \int_use:N \l_@@_initial_i_int -
2286         \int_use:N \l_@@_initial_j_int
2287     }
2288     {
2289         \int_add:Nn \l_@@_initial_i_int { #3 }
2290         \int_add:Nn \l_@@_initial_j_int { #4 }
2291         \bool_set_true:N \l_@@_initial_open_bool
2292         \bool_set_true:N \l_@@_stop_loop_bool
2293     }
2294     {
2295         \cs_if_exist:cTF
2296         {
2297             pgf @ sh @ ns @ \@@_env:
2298             - \int_use:N \l_@@_initial_i_int
2299             - \int_use:N \l_@@_initial_j_int
2300         }
2301         { \bool_set_true:N \l_@@_stop_loop_bool }
2302         {
2303             \cs_set:cpn
2304             {
2305                 @@ _ dotted _
2306                 \int_use:N \l_@@_initial_i_int -
2307                 \int_use:N \l_@@_initial_j_int
2308             }
2309             { }
2310         }
2311     }
2312 }
2313 }

```

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.

```

2314     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
2315     {
2316         { \int_use:N \l_@@_initial_i_int }
2317         { \int_use:N \l_@@_initial_j_int }
2318         { \int_use:N \l_@@_final_i_int }
2319         { \int_use:N \l_@@_final_j_int }
2320     }
2321 }

```

```

2322 \cs_new_protected:Npn \@@_set_initial_coords:

```

```

2323 {
2324   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2325   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2326 }
2327 \cs_new_protected:Npn \@@_set_final_coords:
2328 {
2329   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2330   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2331 }
2332 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
2333 {
2334   \pgfpointanchor
2335   {
2336     \@@_env:
2337     - \int_use:N \l_@@_initial_i_int
2338     - \int_use:N \l_@@_initial_j_int
2339   }
2340   { #1 }
2341   \@@_set_initial_coords:
2342 }
2343 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
2344 {
2345   \pgfpointanchor
2346   {
2347     \@@_env:
2348     - \int_use:N \l_@@_final_i_int
2349     - \int_use:N \l_@@_final_j_int
2350   }
2351   { #1 }
2352   \@@_set_final_coords:
2353 }

```

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.

```

2354 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
2355 {
2356   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2357   {
2358     \@@_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.

```

2359   \group_begin:
2360     \int_compare:nNnTF { #1 } = 0
2361     { \color { nicematrix-first-row } }
2362     {

```

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.

```

2363       \int_compare:nNnT { #1 } = \l_@@_last_row_int
2364       { \color { nicematrix-last-row } }
2365     }
2366     \keys_set:nn { NiceMatrix / xdots } { #3 }
2367     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2368     \@@_actually_draw_Ldots:
2369   \group_end:
2370 }
2371 }

```

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`.

```

2372 \cs_new_protected:Npn \@@_actually_draw_Ldots:
2373 {
2374   \bool_if:NTF \l_@@_initial_open_bool
2375   {
2376     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2377     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2378     \dim_add:Nn \l_@@_x_initial_dim \@@_tab_or_array_colsep:
2379     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
2380     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2381   }
2382   { \@@_set_initial_coords_from_anchor:n { base-east } }
2383   \bool_if:NTF \l_@@_final_open_bool
2384   {
2385     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2386     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2387     \dim_sub:Nn \l_@@_x_final_dim \@@_tab_or_array_colsep:
2388     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
2389     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2390   }
2391   { \@@_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 (?).

```

2392   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
2393   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
2394   \@@_draw_line:
2395 }

```

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.

```

2396 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
2397 {
2398   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2399   {
2400     \@@_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.

```

2401   \group_begin:
2402   \int_compare:nNnTF { #1 } = 0
2403   { \color { nicematrix-first-row } }
2404   {

```

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.

```

2405     \int_compare:nNnT { #1 } = \l_@@_last_row_int
2406     { \color { nicematrix-last-row } }
2407   }
2408   \keys_set:nn { NiceMatrix / xdots } { #3 }
2409   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2410   \@@_actually_draw_Cdots:
2411   \group_end:
2412 }
2413 }

```

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.`

```

2414 \cs_new_protected:Npn \@@_actually_draw_Cdots:
2415 {
2416   \bool_if:NTF \l_@@_initial_open_bool
2417   {
2418     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2419     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2420     \dim_add:Nn \l_@@_x_initial_dim
2421     { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
2422   }
2423   { \@@_set_initial_coords_from_anchor:n { mid-east } }
2424   \bool_if:NTF \l_@@_final_open_bool
2425   {
2426     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2427     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2428     \dim_sub:Nn \l_@@_x_final_dim
2429     { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
2430   }
2431   { \@@_set_final_coords_from_anchor:n { mid-west } }
2432   \bool_lazy_and:nnTF
2433   \l_@@_initial_open_bool
2434   \l_@@_final_open_bool
2435   {
2436     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2437     \dim_set_eq:NN \l_tmpa_dim \pgf@y
2438     \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
2439     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
2440     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
2441   }
2442   {
2443     \bool_if:NT \l_@@_initial_open_bool
2444     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
2445     \bool_if:NT \l_@@_final_open_bool
2446     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
2447   }
2448   \@@_draw_line:
2449 }

```

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.

```

2450 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
2451 {
2452   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2453   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2454   {
2455     \@@_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.

```

2456   \group_begin:
2457   \int_compare:nNnTF { #2 } = 0
2458   { \color { nicematrix-first-col } }
2459   {

```

```

2460         \int_compare:nNnT { #2 } = \l_@@_last_col_int
2461         { \color { nicematrix-last-col } }
2462     }
2463     \keys_set:nn { NiceMatrix / xdots } { #3 }
2464     \@@_actually_draw_Vdots:
2465 \group_end:
2466 }
2467 }

```

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`.

```

2468 \cs_new_protected:Npn \@@_actually_draw_Vdots:
2469 {

```

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

```

2470     \bool_set_false:N \l_tmpa_bool
2471     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
2472     {
2473         \@@_set_initial_coords_from_anchor:n { south-west }
2474         \@@_set_final_coords_from_anchor:n { north-west }
2475         \bool_set:Nn \l_tmpa_bool
2476             { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
2477     }

```

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

```

2478     \bool_if:NTF \l_@@_initial_open_bool
2479     {
2480         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2481         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2482     }
2483     { \@@_set_initial_coords_from_anchor:n { south } }
2484     \bool_if:NTF \l_@@_final_open_bool
2485     {
2486         \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2487         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2488     }
2489     { \@@_set_final_coords_from_anchor:n { north } }
2490     \bool_if:NTF \l_@@_initial_open_bool
2491     {
2492         \bool_if:NTF \l_@@_final_open_bool
2493         {
2494             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2495             \dim_set_eq:NN \l_tmpa_dim \pgf@x
2496             \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
2497             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
2498             \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.

```

2499         \int_compare:nNnT \l_@@_last_col_int > { -2 }
2500         {

```

```

2501         \int_compare:nNt \l_@@_initial_j_int = \g_@@_col_total_int
2502         {
2503             \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
2504             \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
2505             \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
2506             \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
2507         }
2508     }
2509 }
2510 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
2511 }
2512 {
2513     \bool_if:NTF \l_@@_final_open_bool
2514     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
2515     {

```

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.

```

2516         \dim_compare:nNf \l_@@_x_initial_dim = \l_@@_x_final_dim
2517         {
2518             \dim_set:Nn \l_@@_x_initial_dim
2519             {
2520                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
2521                 \l_@@_x_initial_dim \l_@@_x_final_dim
2522             }
2523             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
2524         }
2525     }
2526 }
2527 \@@_draw_line:
2528 }

```

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.

```

2529 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
2530 {
2531     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2532     {
2533         \@@_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.

```

2534     \group_begin:
2535     \keys_set:nn { NiceMatrix / xdots } { #3 }
2536     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2537     \@@_actually_draw_Ddots:
2538     \group_end:
2539 }
2540 }

```

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`.

```

2541 \cs_new_protected:Npn \@@_actually_draw_Ddots:
2542 {
2543   \bool_if:NTF \l_@@_initial_open_bool
2544   {
2545     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2546     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2547     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2548     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2549   }
2550   { \@@_set_initial_coords_from_anchor:n { south-east } }
2551   \bool_if:NTF \l_@@_final_open_bool
2552   {
2553     \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2554     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2555     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2556     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2557   }
2558   { \@@_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.

```

2559   \bool_if:NT \l_@@_parallelize_diags_bool
2560   {
2561     \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).

```

2562     \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.

```

2563     {
2564       \dim_gset:Nn \g_@@_delta_x_one_dim
2565       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
2566       \dim_gset:Nn \g_@@_delta_y_one_dim
2567       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
2568     }

```

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`.

```

2569     {
2570       \dim_set:Nn \l_@@_y_final_dim
2571       {
2572         \l_@@_y_initial_dim +
2573         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2574         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
2575       }
2576     }
2577   }
2578   \@@_draw_line:
2579 }

```

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.

```

2580 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
2581 {
2582   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2583   {
2584     \@@_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.

```

2585     \group_begin:
2586     \keys_set:nn { NiceMatrix / xdots } { #3 }
2587     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2588     \@@_actually_draw_Iddots:
2589     \group_end:
2590   }
2591 }

```

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.`

```

2592 \cs_new_protected:Npn \@@_actually_draw_Iddots:
2593 {
2594   \bool_if:NTF \l_@@_initial_open_bool
2595   {
2596     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2597     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2598     \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
2599     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2600   }
2601   { \@@_set_initial_coords_from_anchor:n { south-west } }
2602   \bool_if:NTF \l_@@_final_open_bool
2603   {
2604     \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
2605     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2606     \@@_qpoint:n { col - \int_use:N \l_@@_final_j_int }
2607     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2608   }
2609   { \@@_set_final_coords_from_anchor:n { north-east } }
2610   \bool_if:NT \l_@@_parallelize_diags_bool
2611   {
2612     \int_gincr:N \g_@@_iddots_int
2613     \int_compare:nNnTF \g_@@_iddots_int = 1
2614     {
2615       \dim_gset:Nn \g_@@_delta_x_two_dim
2616       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
2617       \dim_gset:Nn \g_@@_delta_y_two_dim
2618       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
2619     }
2620     {
2621       \dim_set:Nn \l_@@_y_final_dim
2622       {
2623         \l_@@_y_initial_dim +
2624         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2625         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
2626       }
2627     }
2628   }
2629   \@@_draw_line:
2630 }

```

The actual instructions for drawing the dotted line with Tikz

The command `\l_@@_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`

```

2631 \cs_new_protected:Npn \l_@@_draw_line:
2632 {
2633   \pgfrememberpicturepositiononpagetrue
2634   \pgf@relevantforpicturesizefalse
2635   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
2636     \l_@@_draw_standard_dotted_line:
2637     \l_@@_draw_non_standard_dotted_line:
2638 }

```

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.

```

2639 \cs_new_protected:Npn \l_@@_draw_non_standard_dotted_line:
2640 {
2641   \begin { scope }
2642   \exp_args:No \l_@@_draw_non_standard_dotted_line:n
2643     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
2644 }

```

We have used the fact that, in PGF, a 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 `\l_@@_draw_non_standard_dotted_line:n` is, in fact, the list of options.

```

2645 \cs_new_protected:Npn \l_@@_draw_non_standard_dotted_line:n #1
2646 {
2647   \draw
2648   [
2649     #1 ,
2650     shorten-> = \l_@@_xdots_shorten_dim ,
2651     shorten-< = \l_@@_xdots_shorten_dim ,
2652   ]
2653     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
2654     -- node [ sloped , above ]
2655       { \c_math_toggle_token \scriptstyle \l_@@_xdots_up_tl \c_math_toggle_token }
2656       node [ sloped , below ]
2657       {
2658         \c_math_toggle_token
2659         \scriptstyle \l_@@_xdots_down_tl
2660         \c_math_toggle_token
2661       }
2662     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
2663   \end { scope }
2664 }

```

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

```

2665 \cs_new_protected:Npn \l_@@_draw_standard_dotted_line:
2666 {

```

First, we put the labels.

```

2667 \bool_lazy_and:nnF
2668 { \tl_if_empty_p:N \l_@@_xdots_up_tl }
2669 { \tl_if_empty_p:N \l_@@_xdots_down_tl }
2670 {
2671   \pgfscope
2672   \pgftransformshift
2673   {
2674     \pgfpointlineattime { 0.5 }
2675     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
2676     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
2677   }
2678   \pgftransformrotate
2679   {
2680     \fp_eval:n
2681     {
2682       atand
2683       (
2684         \l_@@_y_final_dim - \l_@@_y_initial_dim ,
2685         \l_@@_x_final_dim - \l_@@_x_initial_dim
2686       )
2687     }
2688   }
2689   \pgfnode
2690   { rectangle }
2691   { south }
2692   {
2693     \c_math_toggle_token
2694     \scriptstyle \l_@@_xdots_up_tl
2695     \c_math_toggle_token
2696   }
2697   { }
2698   { \pgfusepath { } }
2699   \pgfnode
2700   { rectangle }
2701   { north }
2702   {
2703     \c_math_toggle_token
2704     \scriptstyle \l_@@_xdots_down_tl
2705     \c_math_toggle_token
2706   }
2707   { }
2708   { \pgfusepath { } }
2709   \endpgfscope
2710 }
2711 \pgfrememberpicturepositiononpagetrue
2712 \pgf@relevantforpicturesizefalse
2713 \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.

```

2714 \dim_zero_new:N \l_@@_l_dim
2715 \dim_set:Nn \l_@@_l_dim
2716 {
2717   \fp_to_dim:n
2718   {
2719     sqrt
2720     (
2721       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
2722       +
2723       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
2724     )
2725   }
2726 }

```

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.

```

2727     \bool_lazy_or:nnF
2728     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
2729     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
2730     \@@_draw_standard_dotted_line_i:
2731     \group_end:
2732 }
2733 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
2734 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
2735 {

```

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

```

2736     \bool_if:NTF \l_@@_initial_open_bool
2737     {
2738         \bool_if:NTF \l_@@_final_open_bool
2739         {
2740             \int_set:Nn \l_tmpa_int
2741             { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
2742         }
2743         {
2744             \int_set:Nn \l_tmpa_int
2745             {
2746                 \dim_ratio:nn
2747                 { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
2748                 \l_@@_inter_dots_dim
2749             }
2750         }
2751     }
2752     {
2753         \bool_if:NTF \l_@@_final_open_bool
2754         {
2755             \int_set:Nn \l_tmpa_int
2756             {
2757                 \dim_ratio:nn
2758                 { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
2759                 \l_@@_inter_dots_dim
2760             }
2761         }
2762         {
2763             \int_set:Nn \l_tmpa_int
2764             {
2765                 \dim_ratio:nn
2766                 { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
2767                 \l_@@_inter_dots_dim
2768             }
2769         }
2770     }

```

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

```

2771     \dim_set:Nn \l_tmpa_dim
2772     {
2773         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2774         \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
2775     }
2776     \dim_set:Nn \l_tmpb_dim
2777     {
2778         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
2779         \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
2780     }

```

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`.

```

2781 \int_set:Nn \l_tmpb_int
2782 {
2783   \bool_if:NTF \l_@@_initial_open_bool
2784     { \bool_if:NTF \l_@@_final_open_bool 1 0 }
2785     { \bool_if:NTF \l_@@_final_open_bool 2 1 }
2786 }

```

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.

```

2787 \dim_gadd:Nn \l_@@_x_initial_dim
2788 {
2789   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
2790   \dim_ratio:nn
2791   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
2792   { 2 \l_@@_l_dim }
2793   * \l_tmpb_int
2794 }
2795 \dim_gadd:Nn \l_@@_y_initial_dim
2796 {
2797   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
2798   \dim_ratio:nn
2799   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
2800   { 2 \l_@@_l_dim }
2801   * \l_tmpb_int
2802 }
2803 \pgf@relevantforpicturesizefalse
2804 \int_step_inline:nnn 0 \l_tmpa_int
2805 {
2806   \pgfpathcircle
2807   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
2808   { \l_@@_radius_dim }
2809   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
2810   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
2811 }
2812 \pgfusepathqfill
2813 }

```

User commands available in the new environments

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The starred versions of these commands are deprecated since version 3.1 but, as of now, they are still available with an error.

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\AtBeginDocument` and the *arg spec* will be rescanned.

```

2814 \AtBeginDocument
2815 {
2816   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
2817   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

2818 \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
2819 {
2820   \int_compare:nNnTF \c@jCol = 0
2821     { \@@_error:nn { in~first~col } \Ldots }
2822     {
2823       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
2824         { \@@_error:nn { in~last~col } \Ldots }
2825         {
2826           \@@_instruction_of_type:nn { Ldots }
2827           { #1 , down = #2 , up = #3 }
2828         }
2829       }
2830     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ldots }
2831     \bool_gset_true:N \g_@@_empty_cell_bool
2832   }

2833 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
2834 {
2835   \int_compare:nNnTF \c@jCol = 0
2836     { \@@_error:nn { in~first~col } \Cdots }
2837     {
2838       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
2839         { \@@_error:nn { in~last~col } \Cdots }
2840         {
2841           \@@_instruction_of_type:nn { Cdots }
2842           { #1 , down = #2 , up = #3 }
2843         }
2844       }
2845     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_cdots }
2846     \bool_gset_true:N \g_@@_empty_cell_bool
2847   }

2848 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
2849 {
2850   \int_compare:nNnTF \c@iRow = 0
2851     { \@@_error:nn { in~first~row } \Vdots }
2852     {
2853       \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
2854         { \@@_error:nn { in~last~row } \Vdots }
2855         {
2856           \@@_instruction_of_type:nn { Vdots }
2857           { #1 , down = #2 , up = #3 }
2858         }
2859       }
2860     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_vdots }
2861     \bool_gset_true:N \g_@@_empty_cell_bool
2862   }

2863 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
2864 {
2865   \int_case:nnF \c@iRow
2866   {
2867     0 { \@@_error:nn { in~first~row } \Ddots }
2868     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
2869   }
2870   {
2871     \int_case:nnF \c@jCol
2872     {
2873       0 { \@@_error:nn { in~first~col } \Ddots }
2874       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
2875     }

```

```

2876         {
2877             \@@_instruction_of_type:nn { Ddots }
2878             { #1 , down = #2 , up = #3 }
2879         }
2880
2881     }
2882     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ddots }
2883     \bool_gset_true:N \g_@@_empty_cell_bool
2884 }

\exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
{
  \int_case:nnF \c@iRow
  {
    0 { \@@_error:nn { in~first~row } \Iddots }
    \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
  }
  {
    \int_case:nnF \c@jCol
    {
      0 { \@@_error:nn { in~first~col } \Iddots }
      \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
    }
    {
      \@@_instruction_of_type:nn { Iddots }
      { #1 , down = #2 , up = #3 }
    }
  }
}
\bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_iddots }
\bool_gset_true:N \g_@@_empty_cell_bool
}
}

```

End of the \AtBeginDocument.

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

```

2908 \cs_new_protected:Npn \@@_Hspace:
2909 {
2910     \bool_gset_true:N \g_@@_empty_cell_bool
2911     \hspace
2912 }

```

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

```

2913 \cs_set_eq:NN \@@_old_multicolumn \multicolumn
2914 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2915 {
2916     % \begin{macrocode}
2917     % We have to act in expandable way since it will begin by a |\multicolumn|.
2918     % \end{macrocode}
2919     \exp_args:NNe
2920     \@@_old_multicolumn
2921     { #1 }

```

We will have to replace \tl_lower_case:n in the future since \tl_lower_case:n seems to be deprecated.

```

2922 {
2923     \exp_args:Ne \str_case:nn { \tl_lower_case:n { #2 } }
2924     {
2925         l { > \@@_Cell: l < \@@_end_Cell: }
2926         r { > \@@_Cell: r < \@@_end_Cell: }

```



```

2927         c { > \@@_Cell: c < \@@_end_Cell: }
2928         { l | } { > \@@_Cell: l < \@@_end_Cell: | }
2929         { r | } { > \@@_Cell: r < \@@_end_Cell: | }
2930         { c | } { > \@@_Cell: c < \@@_end_Cell: | }
2931         { | l } { | > \@@_Cell: l < \@@_end_Cell: }
2932         { | r } { | > \@@_Cell: r < \@@_end_Cell: }
2933         { | c } { | > \@@_Cell: c < \@@_end_Cell: }
2934         { | l | } { | > \@@_Cell: l < \@@_end_Cell: | }
2935         { | r | } { | > \@@_Cell: r < \@@_end_Cell: | }
2936         { | c | } { | > \@@_Cell: c < \@@_end_Cell: | }
2937     }
2938 }
2939 { #3 }

```

The `\peek_remove_spaces:n` is mandatory.

```

2940 \peek_remove_spaces:n
2941 {
2942     \int_compare:nNnT #1 > 1
2943     {
2944         \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2945         { \int_use:N \c@iRow - \int_use:N \c@jCol }
2946         \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2947         \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2948         {
2949             { \int_use:N \c@iRow }
2950             { \int_use:N \c@jCol }
2951             { \int_use:N \c@iRow }
2952             { \int_eval:n { \c@jCol + #1 - 1 } }
2953         }
2954     }
2955     \int_gadd:Nn \c@jCol { #1 - 1 }
2956 }
2957 }

```

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`.

```

2958 \cs_new:Npn \@@_Hdotsfor:
2959 {
2960     \multicolumn { 1 } { c } { }
2961     \@@_Hdotsfor_i
2962 }

```

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:`).

```

2963 \AtBeginDocument
2964 {
2965     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
2966     \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.

```

2967     \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
2968     {
2969         \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
2970         {
2971             \@@_Hdotsfor:nnnn
2972             { \int_use:N \c@iRow }
2973             { \int_use:N \c@jCol }
2974             { #2 }
2975             {
2976                 #1 , #3 ,

```

```

2977         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
2978     }
2979 }
2980 \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
2981 }
2982 }

```

Enf of \AtBeginDocument.

```

2983 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
2984 {
2985     \bool_set_false:N \l_@@_initial_open_bool
2986     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

2987     \int_set:Nn \l_@@_initial_i_int { #1 }
2988     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

2989     \int_compare:nNnTF #2 = 1
2990     {
2991         \int_set:Nn \l_@@_initial_j_int 1
2992         \bool_set_true:N \l_@@_initial_open_bool
2993     }
2994     {
2995         \cs_if_exist:cTF
2996         {
2997             pgf @ sh @ ns @ \@@_env:
2998             - \int_use:N \l_@@_initial_i_int
2999             - \int_eval:n { #2 - 1 }
3000         }
3001         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
3002         {
3003             \int_set:Nn \l_@@_initial_j_int { #2 }
3004             \bool_set_true:N \l_@@_initial_open_bool
3005         }
3006     }
3007     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
3008     {
3009         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3010         \bool_set_true:N \l_@@_final_open_bool
3011     }
3012     {
3013         \cs_if_exist:cTF
3014         {
3015             pgf @ sh @ ns @ \@@_env:
3016             - \int_use:N \l_@@_final_i_int
3017             - \int_eval:n { #2 + #3 }
3018         }
3019         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
3020         {
3021             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3022             \bool_set_true:N \l_@@_final_open_bool
3023         }
3024     }
3025     \group_begin:
3026     \int_compare:nNnTF { #1 } = 0
3027     { \color { nicematrix-first-row } }
3028     {
3029         \int_compare:nNnT { #1 } = \g_@@_row_total_int
3030         { \color { nicematrix-last-row } }
3031     }
3032     \keys_set:nn { NiceMatrix / xdots } { #4 }
3033     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }

```

```

3034 \@@_actually_draw_Ldots:
3035 \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).

```

3036 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
3037 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
3038 }

3039 \AtBeginDocument
3040 {
3041   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3042   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3043   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
3044   {
3045     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
3046     {
3047       \@@_Vdotsfor:nnnn
3048       { \int_use:N \c@iRow }
3049       { \int_use:N \c@jCol }
3050       { #2 }
3051       {
3052         #1 , #3 ,
3053         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3054       }
3055     }
3056   }
3057 }

```

Enf of \AtBeginDocument.

```

3058 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
3059 {
3060   \bool_set_false:N \l_@@_initial_open_bool
3061   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

3062 \int_set:Nn \l_@@_initial_j_int { #2 }
3063 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

3064 \int_compare:nNnTF #1 = 1
3065 {
3066   \int_set:Nn \l_@@_initial_i_int 1
3067   \bool_set_true:N \l_@@_initial_open_bool
3068 }
3069 {
3070   \cs_if_exist:cTF
3071   {
3072     pgf @ sh @ ns @ \@@_env:
3073     - \int_eval:n { #1 - 1 }
3074     - \int_use:N \l_@@_initial_j_int
3075   }
3076   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
3077   {
3078     \int_set:Nn \l_@@_initial_i_int { #1 }
3079     \bool_set_true:N \l_@@_initial_open_bool
3080   }
3081 }
3082 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
3083 {
3084   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3085   \bool_set_true:N \l_@@_final_open_bool

```

```

3086     }
3087     {
3088         \cs_if_exist:cTF
3089         {
3090             pgf @ sh @ ns @ \@@_env:
3091             - \int_eval:n { #1 + #3 }
3092             - \int_use:N \l_@@_final_j_int
3093         }
3094         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
3095         {
3096             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3097             \bool_set_true:N \l_@@_final_open_bool
3098         }
3099     }
3100     \group_begin:
3101     \int_compare:nNnTF { #2 } = 0
3102     { \color { nicematrix-first-col } }
3103     {
3104         \int_compare:nNnT { #2 } = \g_@@_col_total_int
3105         { \color { nicematrix-last-col } }
3106     }
3107     \keys_set:nn { NiceMatrix / xdots } { #4 }
3108     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3109     \@@_actually_draw_Vdots:
3110     \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).

```

3111     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
3112     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
3113 }

```

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`.

```

3114 \cs_new_protected:Npn \@@_rotate:
3115 {
3116     \bool_if:NTF \l_@@_NiceTabular_bool
3117     { \group_insert_after:N \@@_rotate_ii: }
3118     { \group_insert_after:N \@@_rotate_i: }
3119 }
3120 \cs_new_protected:Npn \@@_rotate_i: { \group_insert_after:N \@@_rotate_ii: }
3121 \cs_new_protected:Npn \@@_rotate_ii: { \group_insert_after:N \@@_rotate_iii: }
3122 \cs_new_protected:Npn \@@_rotate_iii:
3123 {
3124     \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.

```

3125     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
3126     {
3127         \vbox_set_top:Nn \l_@@_cell_box
3128         {
3129             \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`).

```

3130         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
3131         \box_use:N \l_@@_cell_box

```

```

3132     }
3133   }
3134 }

```

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).⁴³

```

3135 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
3136   { \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).

```

3137 \AtBeginDocument
3138 {
3139   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
3140   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3141   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
3142     {
3143       \group_begin:
3144       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
3145       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3146       \use:e
3147       {
3148         \@@_line_i:nn
3149         { \@@_double_int_eval:n #2 \q_stop }
3150         { \@@_double_int_eval:n #3 \q_stop }
3151       }
3152       \group_end:
3153     }
3154 }
3155 \cs_new_protected:Npn \@@_line_i:nn #1 #2
3156 {
3157   \bool_set_false:N \l_@@_initial_open_bool
3158   \bool_set_false:N \l_@@_final_open_bool
3159   \bool_if:nTF
3160     {
3161       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
3162       ||
3163       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
3164     }
3165     {
3166       \@@_error:nnn { unknown~cell~for~line~in~code~after } { #1 } { #2 }
3167     }
3168     { \@@_draw_line_ii:nn { #1 } { #2 } }
3169 }
3170 \AtBeginDocument
3171 {
3172   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
3173   {

```

⁴³Indeed, we want that the user may use the command `\line` in `code-after` with LaTeX counters in the arguments — with the command `\value`.

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:`.

```

3174     \c_@@_pgfortikzpicture_tl
3175     \@@_draw_line_iii:nn { #1 } { #2 }
3176     \c_@@_endpgfortikzpicture_tl
3177   }
3178 }

```

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

```

3179 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
3180 {
3181   \pgfrememberpicturepositiononpagetrue
3182   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
3183   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3184   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3185   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
3186   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3187   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3188   \@@_draw_line:
3189 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

Colors of cells, rows and columns

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

```

3190 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
3191 {
3192   \tl_set:Nn \l_tmpa_tl { #1 }
3193   \tl_set:Nn \l_tmpb_tl { #2 }
3194 }

```

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

```

3195 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
3196 {
3197   \tl_if_blank:nF { #2 }
3198   {
3199     \pgfpicture
3200     \pgf@relevantforpicturesizefalse
3201     \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }

```

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

```

3202     \@@_qpoint:n { col - 1 }
3203     \int_compare:nNnTF \l_@@_first_col_int = 0
3204       { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
3205       { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
3206     \@@_qpoint:n { col - \@@_succ:n \c@jCol }
3207     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
3208     \clist_map_inline:nn { #3 }
3209     {
3210       \tl_set:Nn \l_tmpa_tl { ##1 }
3211       \tl_if_in:NnTF \l_tmpa_tl { - }
3212         { \@@_cut_on_hyphen:w ##1 \q_stop }
3213         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
3214       \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
3215       \tl_if_empty:NT \l_tmpb_tl
3216         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

```

3217         \int_compare:nNnT \l_tmpb_tl > \c@iRow
3218         { \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`.

```

3219         @@_qpoint:n { row - @@_succ:n \l_tmpb_tl }
3220         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
3221         @@_qpoint:n { row - \l_tmpa_tl }
3222         \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
3223         \pgfpathrectanglecorners
3224         { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
3225         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3226     }
3227     \pgfusepathqfill
3228     \endpgfpicture
3229 }
3230 }

```

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

```

3231 \NewDocumentCommand \@@_columncolor { 0 { } m m }
3232 {
3233     \tl_if_blank:nF { #2 }
3234     {
3235         \pgfpicture
3236         \pgf@relevantforpicturesizefalse
3237         \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
3238         @@_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.

```

3239         \dim_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
3240         @@_qpoint:n { row - @@_succ:n \c@iRow }
3241         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
3242         \clist_map_inline:nn { #3 }
3243         {
3244             \tl_set:Nn \l_tmpa_tl { ##1 }
3245             \tl_if_in:NnTF \l_tmpa_tl { - }
3246             { \@@_cut_on_hyphen:w ##1 \q_stop }
3247             { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
3248             \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
3249             \tl_if_empty:NT \l_tmpb_tl
3250             { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
3251             \int_compare:nNnT \l_tmpb_tl > \c@jCol
3252             { \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`.

```

3253         @@_qpoint:n { col - \l_tmpa_tl }
3254         \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
3255         { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
3256         { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
3257         @@_qpoint:n { col - @@_succ:n \l_tmpb_tl }
3258         \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
3259         \pgfpathrectanglecorners
3260         { \pgfpoint \l_tmpc_dim \l_tmpa_dim }
3261         { \pgfpoint \l_tmpd_dim \l_tmpb_dim }
3262     }
3263     \pgfusepathqfill
3264     \endpgfpicture
3265 }
3266 }

```

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

```

3267 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
3268 {
3269     \tl_if_blank:nF { #2 }

```

```

3270 {
3271   \pgfpicture
3272   \pgf@relevantforpicturesizefalse
3273   \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
3274   \clist_map_inline:nn { #3 }
3275   {
3276     \@@_cut_on_hyphen:w ##1 \q_stop
3277     \@@_qpoint:n { row - \l_tmpa_tl }
3278     \bool_lazy_and:nnT
3279       { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
3280       { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
3281     {
3282       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
3283       \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
3284       \dim_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
3285       \@@_qpoint:n { col - \l_tmpb_tl }
3286       \int_compare:nNnTF \l_@@_first_col_int = \l_tmpb_tl
3287         { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
3288         { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
3289       \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
3290       \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
3291       \pgfpathrectanglecorners
3292         { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
3293         { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
3294     }
3295   }
3296   \pgfusepathqfill
3297   \endpgfpicture
3298 }
3299 }

```

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

```

3300 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
3301 {
3302   \tl_if_blank:nF { #2 }
3303   {
3304     \pgfpicture
3305     \pgf@relevantforpicturesizefalse
3306     \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
3307     \@@_cut_on_hyphen:w #3 \q_stop
3308     \bool_lazy_and:nnT
3309       { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
3310       { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
3311     {
3312       \@@_qpoint:n { row - \l_tmpa_tl }
3313       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
3314       \@@_qpoint:n { col - \l_tmpb_tl }
3315       \int_compare:nNnTF \l_@@_first_col_int = \l_tmpb_tl
3316         { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
3317         { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
3318       \@@_cut_on_hyphen:w #4 \q_stop
3319       \int_compare:nNnT \l_tmpa_tl > \c@iRow
3320         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
3321       \int_compare:nNnT \l_tmpb_tl > \c@jCol
3322         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
3323       \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
3324       \dim_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
3325       \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
3326       \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
3327       \pgfpathrectanglecorners
3328         { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
3329         { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
3330       \pgfusepathqfill

```



```

3331     }
3332     \endpgfpicture
3333 }
3334 }

```

The command `\rowcolors` (accessible in the `code-before`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`. Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

The last optional argument is for options. As of now, there is only one key available : `respect-blocks`.

```

3335 \keys_define:nn { NiceMatrix / rowcolors }
3336 {
3337     respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
3338     respect-blocks .default:n = true ,
3339     unknown .code:n = \@@_error:n { Unknown-option-for-rowcolors }
3340 }

3341 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
3342 {
3343     \keys_set:nn { NiceMatrix / rowcolors } { #5 }
3344     \bool_lazy_and:nnTF
3345         \l_@@_respect_blocks_bool
3346         { \cs_if_exist_p:c { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } }
3347         { \@@_rowcolors_i:nnnn { #1 } { #2 } { #3 } { #4 } }
3348         {
3349             \int_step_inline:nnn { #2 } { \int_use:N \c@iRow }
3350             {
3351                 \int_if_odd:nTF { ##1 }
3352                 { \@@_rowcolor [ #1 ] { #3 } }
3353                 { \@@_rowcolor [ #1 ] { #4 } }
3354                 { ##1 }
3355             }
3356         }
3357     }

3358     \cs_new_protected:Npn \@@_rowcolors_i:nnnn #1 #2 #3 #4
3359     {
3360         \seq_set_eq:Nc \l_tmpb_seq
3361         { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }

```

We don't want to take into account a block which is completely in the “first column” of (number 0) or in the “last column”.

```

3362     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
3363     { \@@_not_in_exterior_p:nnnn ##1 }

```

The counter `\l_tmpa_int` will be the index of the loop.

```

3364     \int_set:Nn \l_tmpa_int { #2 }

```

The boolean `\l_tmpa_bool` will indicate whereas we are in a row of the first color or of the second color.

```

3365     \bool_set_false:N \l_tmpa_bool

```

We recall that, in the `code-before`, `\c@iRow` is the total number of rows of the array (excepted the potential exterior rows).

```

3366     \int_do_until:nNnn \l_tmpa_int > \c@iRow
3367     {
3368         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
3369         { \@@_intersect_our_row_p:nnnn ##1 }

```

We compute in `\l_tmpb_int` the last row covered by a block.

```

3370         \int_set_eq:NN \l_tmpb_int \l_tmpa_int
3371         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_ii:nnnn ##1 }
3372         \bool_if:NTF \l_tmpa_bool
3373         {

```

```

3374         \@@_rowcolor [ #1 ] { #4 }
3375         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
3376         \bool_set_false:N \l_tmpa_bool
3377     }
3378     {
3379         \@@_rowcolor [ #1 ] { #3 }
3380         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
3381         \bool_set_true:N \l_tmpa_bool
3382     }
3383     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
3384 }
3385 }

3386 \cs_new_protected:Npn \@@_rowcolors_ii:nnnn #1 #2 #3 #4
3387 {
3388     \int_compare:nNnT { #3 } > \l_tmpb_int
3389     { \int_set:Nn \l_tmpb_int { #3 } }
3390 }

3391 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnn p
3392 {
3393     \bool_lazy_or:nnTF
3394     { \int_compare_p:nNn { #4 } = \c_zero_int }
3395     { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
3396     \prg_return_false:
3397     \prg_return_true:
3398 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

3399 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnn p
3400 {
3401     \bool_if:nTF
3402     {
3403         \int_compare_p:n { #1 <= \l_tmpa_int }
3404         &&
3405         \int_compare_p:n { \l_tmpa_int <= #3 }
3406     }
3407     \prg_return_true:
3408     \prg_return_false:
3409 }

3410 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
3411 {
3412     \int_step_inline:nn { \int_use:N \c@iRow }
3413     {
3414         \int_step_inline:nn { \int_use:N \c@jCol }
3415         {
3416             \int_if_even:nTF { ####1 + ##1 }
3417             { \@@_cellcolor [ #1 ] { #2 } }
3418             { \@@_cellcolor [ #1 ] { #3 } }
3419             { ##1 - ####1 }
3420         }
3421     }
3422 }

```

When the user uses the key colortbl-like, the following command will be linked to \cellcolor in the tabular.

```

3423 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
3424 {
3425     \tl_gput_right:Nx \g_nicematrix_code_before_tl
3426     { \cellcolor [ #1 ] { #2 } { \int_use:N \c@iRow - \int_use:N \c@jCol } }
3427 }

```

When the user uses the key `rowcolor-in-tabular`, the following command will be linked to `\rowcolor` in the tabular.

```

3428 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
3429 {
3430   \tl_gput_right:Nx \g_nicematrix_code_before_tl
3431   {
3432     \exp_not:N \rectanglecolor [ #1 ] { #2 }
3433     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3434     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
3435   }
3436 }

```

```

3437 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
3438 {
3439   \int_compare:nNnT \c@iRow = 1
3440   {

```

You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells).

```

3441     \tl_gput_left:Nx \g_nicematrix_code_before_tl
3442     { \exp_not:N \columncolor [ #1 ] { #2 } { \int_use:N \c@jCol } }
3443   }
3444 }

```

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`.

```

3445 \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`.

```

3446 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
3447 {
3448   \int_compare:nNnTF \l_@@_first_col_int = 0
3449   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
3450   {
3451     \int_compare:nNnTF \c@jCol = 0
3452     {
3453       \int_compare:nNnF \c@iRow = { -1 }
3454       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
3455     }
3456     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
3457   }
3458 }

```

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.

```

3459 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
3460 {
3461   \int_compare:nNnF \c@iRow = 0

```

```

3462     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
3463   }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the column #1. #2 is the number of consecutive occurrences of |.

```

3464 \cs_new_protected:Npn \@@_vline:nn #1 #2
3465 {

```

The following test is for the case where the user don't use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

3466   \int_compare:nNnT { #1 } < { \c@jCol + 2 }
3467   {
3468     \pgfpicture
3469     \@@_vline_i:nn { #1 } { #2 }
3470     \endpgfpicture
3471   }
3472 }
3473 \cs_new_protected:Npn \@@_vline_i:nn #1 #2
3474 {

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column.

```

3475   \tl_set:Nx \l_tmpb_tl { #1 }
3476   \int_step_variable:nNn \c@iRow \l_tmpa_tl
3477   {

```

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.

```

3478     \bool_gset_true:N \g_tmpa_bool
3479     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3480     { \@@_test_if_vline_in_block:nnnn ##1 }
3481     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
3482     { \@@_test_if_vline_in_block:nnnn ##1 }
3483     \clist_if_empty:NF \l_@@_except_corners_clist
3484     \@@_test_in_corner_v:
3485     \bool_if:NT \g_tmpa_bool
3486     { \@@_vline_ii:nnnn { #1 } { #2 } \l_tmpa_tl \l_tmpb_tl }
3487   }
3488 }

```

```

3489 \cs_new_protected:Npn \@@_test_in_corner_v:
3490 {
3491   \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
3492   {
3493     \seq_if_in:NxT
3494     \l_@@_empty_corner_cells_seq
3495     { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
3496     { \bool_set_false:N \g_tmpa_bool }
3497   }
3498   {
3499     \seq_if_in:NxT
3500     \l_@@_empty_corner_cells_seq
3501     { \l_tmpa_tl - \l_tmpb_tl }
3502     {
3503       \int_compare:nNnTF \l_tmpb_tl = 1
3504       { \bool_set_false:N \g_tmpa_bool }
3505       {
3506         \seq_if_in:NxT
3507         \l_@@_empty_corner_cells_seq

```

```

3508         { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
3509         { \bool_set_false:N \g_tmpa_bool }
3510     }
3511 }
3512 }
3513 }

```

#1 is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the number of the rows between which the rule has to be drawn.

```

3514 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
3515 {
3516     \pgfrememberpicturepositiononpagetrue
3517     \pgf@relevantforpicturesizefalse
3518     \@@_qpoint:n { row - #3 }
3519     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3520     \@@_qpoint:n { col - #1 }
3521     \dim_set_eq:NN \l_tmpb_dim \pgf@x
3522     \@@_qpoint:n { row - \@@_succ:n { #4 } }
3523     \dim_set_eq:NN \l_tmpc_dim \pgf@y
3524     \bool_lazy_and:nnT
3525     { \int_compare_p:nNn { #2 } > 1 }
3526     { ! \tl_if_blank_p:V \CT@drsc@ }
3527     {
3528         \group_begin:
3529         \CT@drsc@
3530         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
3531         \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
3532         \dim_set:Nn \l_tmpd_dim
3533         { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
3534         \pgfpathrectanglecorners
3535         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
3536         { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
3537         \pgfusepathqfill
3538         \group_end:
3539     }
3540     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
3541     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
3542     \prg_replicate:nn { #2 - 1 }
3543     {
3544         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
3545         \dim_sub:Nn \l_tmpb_dim \doublerulesep
3546         \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
3547         \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
3548     }
3549     \CT@arc@
3550     \pgfsetlinewidth { 1.1 \arrayrulewidth }
3551     \pgfsetrectcap
3552     \pgfusepathqstroke
3553 }

```

The following draws a complete vertical rule in the column #1 (#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key `except-corners` is not used).

```

3554 \cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
3555 { \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

```

The command `\@@_draw_hlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `except-corners` is used).

```

3556 \cs_new_protected:Npn \@@_draw_vlines:
3557 {

```

```

3558 \int_step_inline:nnn
3559 { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
3560 { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
3561 { \@@_vline:nn { ##1 } 1 }
3562 }

```

The horizontal rules

The following command will be executed in the `internal-code-after`. The row will be drawn *before* the row #1. #2 is the number of consecutive occurrences of `\Hline`.

```

3563 \cs_new_protected:Npn \@@_hline:nn #1 #2
3564 {
3565   \pgfpicture
3566   \@@_hline_i:nn { #1 } { #2 }
3567   \endpgfpicture
3568 }

3569 \cs_new_protected:Npn \@@_hline_i:nn #1 #2
3570 {

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column.

```

3571   \tl_set:Nn \l_tmpa_tl { #1 }
3572   \int_step_variable:nNn \c@jCol \l_tmpb_tl
3573   {

```

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.

```

3574     \bool_gset_true:N \g_tmpa_bool
3575     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3576     { \@@_test_if_hline_in_block:nnnn ##1 }
3577     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
3578     { \@@_test_if_hline_in_block:nnnn ##1 }
3579     \clist_if_empty:NF \l_@@_except_corners_clist \@@_test_in_corner_h:
3580     \bool_if:NT \g_tmpa_bool
3581     { \@@_hline_ii:nnnn { #1 } { #2 } \l_tmpb_tl \l_tmpb_tl }
3582   }
3583 }

```

```

3584 \cs_new_protected:Npn \@@_test_in_corner_h:
3585 {
3586   \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
3587   {
3588     \seq_if_in:NxT
3589     \l_@@_empty_corner_cells_seq
3590     { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
3591     { \bool_set_false:N \g_tmpa_bool }
3592   }
3593   {
3594     \seq_if_in:NxT
3595     \l_@@_empty_corner_cells_seq
3596     { \l_tmpa_tl - \l_tmpb_tl }
3597     {
3598       \int_compare:nNnTF \l_tmpa_tl = 1
3599       { \bool_set_false:N \g_tmpa_bool }
3600       {
3601         \seq_if_in:NxT
3602         \l_@@_empty_corner_cells_seq
3603         { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
3604         { \bool_set_false:N \g_tmpa_bool }

```

```

3605     }
3606   }
3607 }
3608 }

```

#1 is the number of the row; #2 is the number of horizontal rules to draw (with potentially a color between); #3 and #4 are the number of the columns between which the rule has to be drawn.

```

3609 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
3610 {
3611   \pgfrememberpicturepositiononpagetrue
3612   \pgf@relevantforpicturesizefalse
3613   \@@_qpoint:n { col - #3 }
3614   \dim_set_eq:NN \l_tmpa_dim \pgf@x
3615   \@@_qpoint:n { row - #1 }
3616   \dim_set_eq:NN \l_tmpb_dim \pgf@y
3617   \@@_qpoint:n { col - \@@_succ:n { #4 } }
3618   \dim_set_eq:NN \l_tmpc_dim \pgf@x
3619   \bool_lazy_and:nnT
3620     { \int_compare_p:nNn { #2 } > 1 }
3621     { ! \tl_if_blank_p:V \CT@drsc@ }
3622     {
3623       \group_begin:
3624       \CT@drsc@
3625       \dim_set:Nn \l_tmpd_dim
3626         { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
3627       \pgfpathrectanglecorners
3628         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3629         { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
3630       \pgfusepathqfill
3631       \group_end:
3632     }
3633   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3634   \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
3635   \prg_replicate:nn { #2 - 1 }
3636   {
3637     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
3638     \dim_sub:Nn \l_tmpb_dim \doublerulesep
3639     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3640     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
3641   }
3642   \CT@arc@
3643   \pgfsetlinewidth { 1.1 \arrayrulewidth }
3644   \pgfsetrectcap
3645   \pgfusepathqstroke
3646 }

3647 \cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
3648 { \@@_hline_ii:nnnn #1 #2 1 { \int_use:N \c@jCol } }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as `\Cdots` and in the corners (if the key `except-corners` is used).

```

3649 \cs_new_protected:Npn \@@_draw_hlines:
3650 {
3651   \int_step_inline:nnn
3652     { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
3653     { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
3654     { \@@_hline:nn { ##1 } 1 }
3655 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

3656 \cs_set:Npn \@@_Hline: { \noalign { \ifnum 0 = ` } \fi \@@_Hline_i:n { 1 } }

```

The argument of the command `\@@_Hline_i:n` is the number of successive `\Hline` found.

```

3657 \cs_set:Npn \@@_Hline_i:n #1
3658 {
3659   \peek_meaning_ignore_spaces:NTF \Hline
3660   { \@@_Hline_ii:nn { #1 + 1 } }
3661   { \@@_Hline_iii:n { #1 } }
3662 }
3663 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
3664 \cs_set:Npn \@@_Hline_iii:n #1
3665 {
3666   \skip_vertical:n
3667   {
3668     \arrayrulewidth * ( #1 )
3669     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
3670   }
3671   \tl_gput_right:Nx \g_@@_internal_code_after_tl
3672   { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
3673   \ifnum 0 = `{ \fi }
3674 }

```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the col) would provide an horizontal rule towards the right in the block delimited by the four arguments `#1`, `#2`, `#3` and `#4`. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to false.

```

3675 \cs_new_protected:Npn \@@_test_if_hline_in_block:nnnn #1 #2 #3 #4
3676 {
3677   \bool_lazy_all:nT
3678   {
3679     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
3680     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
3681     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
3682     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
3683   }
3684   { \bool_gset_false:N \g_tmpa_bool }
3685 }

```

The same for vertical rules.

```

3686 \cs_new_protected:Npn \@@_test_if_vline_in_block:nnnn #1 #2 #3 #4
3687 {
3688   \bool_lazy_all:nT
3689   {
3690     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
3691     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
3692     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
3693     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
3694   }
3695   { \bool_gset_false:N \g_tmpa_bool }
3696 }

```

The key except-corners

When the key `except-corners` is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

```

3697 \cs_new_protected:Npn \@@_compute_corners:
3698 {

```


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

```

3699 \seq_clear_new:N \l_@@_empty_corner_cells_seq
3700 \clist_map_inline:Nn \l_@@_except_corners_clist
3701 {
3702   \str_case:nnF { ##1 }
3703   {
3704     { NW }
3705     { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
3706     { NE }
3707     { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
3708     { SW }
3709     { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
3710     { SE }
3711     { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
3712   }
3713   { \@@_error:nn { bad~corner } { ##1 } }
3714 }
3715 }
```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_empty_corner_cells_seq`.

The six arguments of `\@@_compute_a_corner:nnnnnn` are as follow:

- #1 and #2 are the number of row and column of the cell which is actually in the corner;
- #3 and #4 are the steps in rows and the step in columns when moving from the corner;
- #5 is the number of the final row when scanning the rows from the corner;
- #6 is the number of the final column when scanning the columns from the corner.

```

3716 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
3717 {
```

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.

```

3718 \bool_set_false:N \l_tmpa_bool
3719 \int_zero_new:N \l_@@_last_empty_row_int
3720 \int_set:Nn \l_@@_last_empty_row_int { #1 }
3721 \int_step_inline:nnnn { #1 } { #3 } { #5 }
3722 {
3723   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
3724   \bool_lazy_or:nnTF
3725   {
3726     \cs_if_exist_p:c
3727     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
3728   }
3729   \l_tmpb_bool
3730   { \bool_set_true:N \l_tmpa_bool }
3731   {
3732     \bool_if:NF \l_tmpa_bool
3733     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
3734   }
3735 }
```

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

```

3736 \bool_set_false:N \l_tmpa_bool
3737 \int_zero_new:N \l_@@_last_empty_column_int
3738 \int_set:Nn \l_@@_last_empty_column_int { #2 }
3739 \int_step_inline:nnnn { #2 } { #4 } { #6 }
```

```

3740 {
3741   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
3742   \bool_lazy_or:nnTF
3743     \l_tmpb_bool
3744     {
3745       \cs_if_exist_p:c
3746         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
3747     }
3748     { \bool_set_true:N \l_tmpa_bool }
3749     {
3750       \bool_if:NF \l_tmpa_bool
3751       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
3752     }
3753 }

```

Now, we loop over the rows.

```

3754   \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
3755   {

```

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

```

3756     \bool_set_false:N \l_tmpa_bool
3757     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
3758     {
3759       \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
3760       \bool_lazy_or:nnTF
3761         \l_tmpb_bool
3762         {
3763           \cs_if_exist_p:c
3764             { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
3765         }
3766         { \bool_set_true:N \l_tmpa_bool }
3767         {
3768           \bool_if:NF \l_tmpa_bool
3769           {
3770             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
3771             \seq_put_right:Nn
3772               \l_@@_empty_corner_cells_seq
3773               { ##1 - #####1 }
3774           }
3775         }
3776     }
3777 }
3778 }

```

The following macro tests whether a cell is in (at least) one of the blocks of the array (or in a cell with a `\diagbox`).

The flag `\l_tmpb_bool` will be raised if the cell #1-#2 is in a block (or in a cell with a `\diagbox`).

```

3779 \cs_new_protected:Npn \@@_test_if_cell_in_block:nn #1 #2
3780 {
3781   \int_set:Nn \l_tmpa_int { #1 }
3782   \int_set:Nn \l_tmpb_int { #2 }
3783   \bool_set_false:N \l_tmpb_bool
3784   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3785     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
3786 }
3787 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6
3788 {
3789   \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
3790   {
3791     \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
3792     {
3793       \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
3794       {

```

```

3795         \int_compare:nNt { #2 } < { \@@_succ:n { #6 } }
3796         { \bool_set_true:N \l_tmpb_bool }
3797     }
3798 }
3799 }
3800 }

```

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`.

```

3801 \cs_new:Npn \@@_hdottedline:
3802 {
3803     \noalign { \skip_vertical:N 2\l_@@_radius_dim }
3804     \@@_hdottedline_i:
3805 }

```

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

```

3806 \cs_new_protected:Npn \@@_hdottedline_i:
3807 {

```

We write in the code-after the instruction that will actually draw the dotted line. It's not possible to draw this dotted line now because we don't know the length of the line (we don't even know the number of columns).

```

3808     \tl_gput_right:Nx \g_@@_internal_code_after_tl
3809     { \@@_hdottedline:n { \int_use:N \c@iRow } }
3810 }

```

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.

```

3811 \AtBeginDocument
3812 {

```

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

```

3813     \cs_new_protected:Npx \@@_hdottedline:n #1
3814     {
3815         \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
3816         \bool_set_true:N \exp_not:N \l_@@_final_open_bool
3817         \c_@@_pgfortikzpicture_tl
3818         \@@_hdottedline_i:n { #1 }
3819         \c_@@_endpgfortikzpicture_tl
3820     }
3821 }

```

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

```

3822 \cs_new_protected:Npn \@@_hdottedline_i:n #1
3823 {
3824     \pgfrememberpicturepositiononpagetrue
3825     \@@_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).

```

3826     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3827     \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3828     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim

```

The dotted line will be extended if the user uses `margin` (or `left-margin` and `right-margin`). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

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

```
3829 \@@_qpoint:n { col - 1 }
3830 \dim_set:Nn \l_@@_x_initial_dim
3831 {
3832   \pgf@x +
3833   \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep
3834   - \l_@@_left_margin_dim
3835 }
3836 \@@_qpoint:n { col - \@@_succ:n \c@jCol }
3837 \dim_set:Nn \l_@@_x_final_dim
3838 {
3839   \pgf@x -
3840   \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep
3841   + \l_@@_right_margin_dim
3842 }
```

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

```
3843 \tl_set:Nn \l_tmpa_tl { ( }
3844 \tl_if_eq:NNF \l_@@_left_delim_tl \l_tmpa_tl
3845 { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
3846 \tl_set:Nn \l_tmpa_tl { ) }
3847 \tl_if_eq:NNF \l_@@_right_delim_tl \l_tmpa_tl
3848 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
```

As of now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```
3849 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
3850 \@@_draw_line:
3851 }
```

Vertical dotted lines

```
3852 \cs_new_protected:Npn \@@_vdottedline:n #1
3853 {
3854   \bool_set_true:N \l_@@_initial_open_bool
3855   \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”.

```
3856 \bool_if:NTF \c_@@_tikz_loaded_bool
3857 {
3858   \tikzpicture
3859   \@@_vdottedline_i:n { #1 }
3860   \endtikzpicture
```

```

3861     }
3862     {
3863         \pgfpicture
3864         \@@_vdottedline_i:n { #1 }
3865         \endpgfpicture
3866     }
3867 }

```

```

3868 \cs_new_protected:Npn \@@_vdottedline_i:n #1
3869 {

```

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.

```

3870     \CT@arc@
3871     \pgfrememberpicturepositiononpagetrue
3872     \@@_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).

```

3873     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
3874     \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
3875     \@@_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.

```

3876     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
3877     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
3878     \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }

```

As of now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

3879     \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
3880     \@@_draw_line:
3881 }

```

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.

```

3882 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

3883 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
3884 {
3885     auto-columns-width .code:n =
3886     {
3887         \bool_set_true:N \l_@@_block_auto_columns_width_bool
3888         \dim_gzero_new:N \g_@@_max_cell_width_dim
3889         \bool_set_true:N \l_@@_auto_columns_width_bool
3890     }
3891 }

```

```

3892 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
3893 {
3894     \int_gincr:N \g_@@_NiceMatrixBlock_int
3895     \dim_zero:N \l_@@_columns_width_dim
3896     \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
3897     \bool_if:NT \l_@@_block_auto_columns_width_bool
3898     {

```

```

3899 \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
3900 {
3901     \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
3902     { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
3903 }
3904 }
3905 }

```

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`).

```

3906 {
3907     \bool_if:NT \l_@@_block_auto_columns_width_bool
3908     {
3909         \iow_shipout:Nn \@mainaux \ExplSyntaxOn
3910         \iow_shipout:Nx \@mainaux
3911         {
3912             \cs_gset:cpn
3913             { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```

3914         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
3915     }
3916     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
3917 }
3918 }

```

The extra nodes

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

```

3919 \cs_generate_variant:Nn \dim_min:nn { v n }
3920 \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`.

```

3921 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
3922 {
3923     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
3924     {
3925         \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
3926         \dim_set_eq:cn { l_@@_row_\@@_i: _min_dim } \c_max_dim
3927         \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
3928         \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }

```

```

3929     }
3930     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
3931     {
3932         \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
3933         \dim_set_eq:cn { l_@@_column_\@@_j: _min_dim } \c_max_dim
3934         \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
3935         \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
3936     }

```

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

```

3937     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
3938     {
3939         \int_step_variable:nnNn
3940         \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.

```

3941         {
3942             \cs_if_exist:cT
3943             { 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`.

```

3944         {
3945             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
3946             \dim_set:cn { l_@@_row_\@@_i: _min_dim }
3947             { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
3948             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
3949             {
3950                 \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
3951                 { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
3952             }

```

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`.

```

3953             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
3954             \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
3955             { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
3956             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
3957             {
3958                 \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
3959                 { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
3960             }
3961         }
3962     }
3963 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

3964     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
3965     {
3966         \dim_compare:nNnT
3967         { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
3968         {
3969             \@@_qpoint:n { row - \@@_i: - base }
3970             \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
3971             \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
3972         }
3973     }
3974     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
3975     {
3976         \dim_compare:nNnT
3977         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
3978         {
3979             \@@_qpoint:n { col - \@@_j: }

```

```

3980         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
3981         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
3982     }
3983 }
3984 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

3985 \cs_new_protected:Npn \@@_create_medium_nodes:
3986 {
3987     \pgfpicture
3988     \pgfrememberpicturepositiononpagetrue
3989     \pgf@relevantforpicturesizefalse
3990     \@@_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”.

```

3991     \tl_set:Nn \l_@@_suffix_tl { -medium }
3992     \@@_create_nodes:
3993     \endpgfpicture
3994 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁴⁴. 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:`.

```

3995 \cs_new_protected:Npn \@@_create_large_nodes:
3996 {
3997     \pgfpicture
3998     \pgfrememberpicturepositiononpagetrue
3999     \pgf@relevantforpicturesizefalse
4000     \@@_computations_for_medium_nodes:
4001     \@@_computations_for_large_nodes:
4002     \tl_set:Nn \l_@@_suffix_tl { - large }
4003     \@@_create_nodes:
4004     \endpgfpicture
4005 }
4006 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
4007 {
4008     \pgfpicture
4009     \pgfrememberpicturepositiononpagetrue
4010     \pgf@relevantforpicturesizefalse
4011     \@@_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”.

```

4012     \tl_set:Nn \l_@@_suffix_tl { - medium }
4013     \@@_create_nodes:
4014     \@@_computations_for_large_nodes:
4015     \tl_set:Nn \l_@@_suffix_tl { - large }
4016     \@@_create_nodes:
4017     \endpgfpicture
4018 }

```

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.

```

4019 \cs_new_protected:Npn \@@_computations_for_large_nodes:
4020 {
4021     \int_set:Nn \l_@@_first_row_int 1
4022     \int_set:Nn \l_@@_first_col_int 1

```

⁴⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

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`.

```

4023 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
4024 {
4025   \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
4026   {
4027     (
4028       \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
4029       \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4030     )
4031     / 2
4032   }
4033   \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4034   { l_@@_row _ \@@_i: _ min _ dim }
4035 }
4036 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
4037 {
4038   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
4039   {
4040     (
4041       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
4042       \dim_use:c
4043       { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4044     )
4045     / 2
4046   }
4047   \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4048   { l_@@_column _ \@@_j: _ max _ dim }
4049 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

4050 \dim_sub:cn
4051 { l_@@_column _ 1 _ min _ dim }
4052 \l_@@_left_margin_dim
4053 \dim_add:cn
4054 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
4055 \l_@@_right_margin_dim
4056 }

```

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).

```

4057 \cs_new_protected:Npn \@@_create_nodes:
4058 {
4059   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4060   {
4061     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4062     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

4063 \@@_pgf_rect_node:nnnnn
4064 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4065 { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
4066 { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
4067 { \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } }
4068 { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
4069 \str_if_empty:NF \l_@@_name_str
4070 {
4071   \pgfnodealias
4072   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }

```

```

4073         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4074     }
4075 }
4076 }

```

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 .

```

4077 \seq_mapthread_function:NNN
4078   \g_@@_multicolumn_cells_seq
4079   \g_@@_multicolumn_sizes_seq
4080   \@@_node_for_multicolumn:nn
4081 }

```

```

4082 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
4083 {
4084   \cs_set:Npn \@@_i: { #1 }
4085   \cs_set:Npn \@@_j: { #2 }
4086 }

```

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”).

```

4087 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
4088 {
4089   \@@_extract_coords_values: #1 \q_stop
4090   \@@_pgf_rect_node:nnnnn
4091     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4092     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
4093     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
4094     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
4095     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
4096   \str_if_empty:NF \l_@@_name_str
4097   {
4098     \pgfnodealias
4099       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4100       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
4101   }
4102 }

```

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?).

```

4103 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
4104 { \@@_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.

```

4105 \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.

```

4106 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
4107 {

```

```

4108 \bool_if:NT \l_@@_NiceTabular_bool
4109 { \tl_if_empty:nF { #4 } { \@@_error:n { angle-option~in~NiceTabular } } }

4110 \tl_set:Nx \l_tmpa_tl
4111 {
4112   { \int_use:N \c@iRow }
4113   { \int_use:N \c@jCol }
4114   { \int_eval:n { \c@iRow + #1 - 1 } }
4115   { \int_eval:n { \c@jCol + #2 - 1 } }
4116 }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components surrounded by curly brackets:

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

We store this information in the sequence `\g_@@_pos_of_blocks_seq`.

```

4117 \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 “object” with six components:

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

```

4118 \seq_gput_left:Nx \g_@@_blocks_seq
4119 {
4120   \l_tmpa_tl
4121   { #3 }
4122   \exp_not:n { { #4 \@@_math_toggle_token: #5 \@@_math_toggle_token: } }
4123 }
4124 }

```

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 of now, it is *not* documented.** Is it really a good idea to provide such a key?

```

4125 \keys_define:nn { NiceMatrix / Block }
4126 {
4127   tikz .tl_set:N = \l_@@_tikz_tl ,
4128   tikz .value_required:n = true ,
4129   color .tl_set:N = \l_@@_color_tl ,
4130   color .value_required:n = true ,
4131 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construction of the array.

```

4132 \cs_new_protected:Npn \@@_draw_blocks:
4133 { \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iii:nnnnn ##1 } }

4134 \cs_new_protected:Npn \@@_Block_iii:nnnnn #1 #2 #3 #4 #5 #6
4135 {

```

The group is for the keys.

```

4136 \group_begin:
4137 \keys_set:nn { NiceMatrix / Block } { #5 }

4138 \tl_if_empty:NF \l_@@_color_tl
4139 {
4140   \tl_gput_right:Nx \g_nicematrix_code_before_tl
4141   {
4142     \exp_not:N \rectanglecolor
4143     { \l_@@_color_tl }
4144     { #1 - #2 }
4145     { #3 - #4 }
4146   }
4147 }

```

```

4148 \cs_set_protected:Npn \diagbox ##1 ##2
4149 {
4150   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4151   {
4152     \@@_actually_diagbox:nnnnnn
4153     { #1 } { #2 } { #3 } { #4 }
4154     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
4155   }
4156 }

4157 \bool_lazy_or:nnTF
4158 { \int_compare_p:nNn { #3 } > \g_@@_row_total_int }
4159 { \int_compare_p:nNn { #4 } > \g_@@_col_total_int }
4160 { \msg_error:nnnn { nicematrix } { Block~too~large } { #1 } { #2 } }
4161 {

```

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.

```

4162 \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	
three	four
six	seven

We highlight the node `1-1-block-short`

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

4163 \pgfpicture
4164 \pgfrememberpicturepositiononpagetrue
4165 \pgf@relevantforpicturesizefalse
4166 \@@_qpoint:n { row - #1 }
4167 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4168 \@@_qpoint:n { col - #2 }
4169 \dim_set_eq:NN \l_tmpb_dim \pgf@x
4170 \@@_qpoint:n { row - \@@_succ:n { #3 } }
4171 \dim_set_eq:NN \l_tmpc_dim \pgf@y
4172 \@@_qpoint:n { col - \@@_succ:n { #4 } }
4173 \dim_set_eq:NN \l_tmpd_dim \pgf@x

```

We construct the node for the block with the name `(#1-#2-block)`.

The function `\@@_pgf_rect_node:nnnnn` takes in as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```

4174 \begin { pgfscope }
4175 \exp_args:Nx \pgfset { \l_@@_tikz_tl }
4176 \@@_pgf_rect_node:nnnnn
4177 { \@@_env: - #1 - #2 - block }
4178 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
4179 \end { pgfscope }

```

We construct the `short` node.

```

4180      \dim_set_eq:NN \l_tmpb_dim \c_max_dim
4181      \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4182      {

```

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.

```

4183          \cs_if_exist:cT
4184          { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
4185          {
4186              \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
4187              \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
4188          }
4189      }

```

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.

```

4190      \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
4191      {
4192          \@@_qpoint:n { col - #2 }
4193          \dim_set_eq:NN \l_tmpb_dim \pgf@x
4194      }
4195      \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
4196      \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4197      {
4198          \cs_if_exist:cT
4199          { pgf @ sh @ ns @ \@@_env: - ##1 - #4 }
4200          {
4201              \pgfpointanchor { \@@_env: - ##1 - #4 } { east }
4202              \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
4203          }
4204      }
4205      \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
4206      {
4207          \@@_qpoint:n { col - \@@_succ:n { #4 } }
4208          \dim_set_eq:NN \l_tmpd_dim \pgf@x
4209      }
4210      \@@_pgf_rect_node:nnnnn
4211      { \@@_env: - #1 - #2 - block - short }
4212      \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 in as arguments the name of the node and two PGF points.

```

4213      \bool_if:NT \l_@@_medium_nodes_bool
4214      {
4215          \@@_pgf_rect_node:nnn
4216          { \@@_env: - #1 - #2 - block - medium }
4217          { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
4218          { \pgfpointanchor { \@@_env: - #3 - #4 - medium } { south-east } }
4219      }

```

Now, we will put the label of the block.

```

4220      \int_compare:nNnTF { #1 } = { #3 }
4221      {

```

We take into account the case of a block of one row in the “first row” or the “end row”.

```

4222          \int_compare:nNnTF { #1 } = 0
4223          { \l_@@_code_for_first_row_tl }
4224          {
4225              \int_compare:nNnT { #1 } = \l_@@_last_row_int
4226              \l_@@_code_for_last_row_tl
4227          }

```

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`.

```
4228 \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }
```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```
4229 \@@_qpoint:n { #1 - #2 - block - short }
```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```
4230 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
4231 \pgfnode { rectangle } { base }
4232 { \box_use_drop:N \l_@@_cell_box } { } { }
4233 }
```

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`).

```
4234 {
4235 \pgftransformshift { \@@_qpoint:n { #1 - #2 - block - short } }
4236 \pgfnode { rectangle } { center }
4237 { \box_use_drop:N \l_@@_cell_box } { } { }
4238 }
4239 \endpgfpicture
4240 }
4241 \group_end:
4242 }
```

How to draw the dotted lines transparently

```
4243 \cs_set_protected:Npn \@@_renew_matrix:
4244 {
4245 \RenewDocumentEnvironment { pmatrix } { } {
4246 { \pNiceMatrix }
4247 { \endpNiceMatrix }
4248 \RenewDocumentEnvironment { vmatrix } { } {
4249 { \vNiceMatrix }
4250 { \endvNiceMatrix }
4251 \RenewDocumentEnvironment { Vmatrix } { } {
4252 { \VNiceMatrix }
4253 { \endVNiceMatrix }
4254 \RenewDocumentEnvironment { bmatrix } { } {
4255 { \bNiceMatrix }
4256 { \endbNiceMatrix }
4257 \RenewDocumentEnvironment { Bmatrix } { } {
4258 { \BNiceMatrix }
4259 { \endBNiceMatrix }
4260 }
```

Automatic arrays

```
4261 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
4262 {
4263 \int_set:Nn \l_@@_nb_rows_int { #1 }
4264 \int_set:Nn \l_@@_nb_cols_int { #2 }
4265 }
4266 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
4267 {
4268 \int_zero_new:N \l_@@_nb_rows_int
4269 \int_zero_new:N \l_@@_nb_cols_int
4270 \@@_set_size:n #4 \q_stop
4271 \begin { NiceArrayWithDelims } { #1 } { #2 }
4272 { * { \l_@@_nb_cols_int } { c } } [ #3 , #5 , #7 ]
```

```

4273 \int_compare:nNnT \l_@@_first_row_int = 0
4274 {
4275     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
4276     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
4277     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
4278 }
4279 \prg_replicate:nn \l_@@_nb_rows_int
4280 {
4281     \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

You put { } before #6 to avoid a hasty expansion of a potential `\arabic{iRow}` at the beginning of the row which would result in an incorrect value of that `iRow` (since `iRow` is incremented in the first cell of the row of the `\halign`).

```

4282     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
4283     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
4284 }
4285 \int_compare:nNnT \l_@@_last_row_int > { -2 }
4286 {
4287     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
4288     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
4289     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
4290 }
4291 \end { NiceArrayWithDelims }
4292 }
4293 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
4294 {
4295     \cs_set_protected:cpn { #1 AutoNiceMatrix }
4296     {
4297         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
4298         \AutoNiceMatrixWithDelims { #2 } { #3 }
4299     }
4300 }
4301 \@@_define_com:nnn p ( )
4302 \@@_define_com:nnn b [ ]
4303 \@@_define_com:nnn v | |
4304 \@@_define_com:nnn V \| \|
4305 \@@_define_com:nnn B \{ \}

```

We define also an command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

4306 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
4307 {
4308     \group_begin:
4309     \bool_set_true:N \l_@@_NiceArray_bool
4310     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
4311     \group_end:
4312 }

```

The redefinition of the command `\dotfill`

```

4313 \cs_set_eq:NN \@@_old_dotfill \dotfill
4314 \cs_new_protected:Npn \@@_dotfill:
4315 {

```

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}`).

```

4316     \@@_old_dotfill
4317     \bool_if:NT \l_@@_NiceTabular_bool
4318     { \group_insert_after:N \@@_dotfill_ii: }
4319     { \group_insert_after:N \@@_dotfill_i: }
4320 }
4321 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
4322 \cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }

```

Now, if the box is not empty (unfortunately, we can't actually test whether the box is empty and that's why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```
4323 \cs_new_protected:Npn \@@_dotfill_iii:
4324 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }
```

The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`.

```
4325 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
4326 {
4327   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4328   {
4329     \@@_actually_diagbox:nnnnnn
4330     { \int_use:N \c_iRow }
4331     { \int_use:N \c_jCol }
4332     { \int_use:N \c_iRow }
4333     { \int_use:N \c_jCol }
4334     { \exp_not:n { #1 } }
4335     { \exp_not:n { #2 } }
4336   }
```

We put the cell with `\diagbox` in the sequence `\g_@@_pos_of_blocks_seq` because a cell with `\diagbox` must be considered as non empty by the key `except-corners`.

```
4337   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
4338   {
4339     { \int_use:N \c_iRow }
4340     { \int_use:N \c_jCol }
4341     { \int_use:N \c_iRow }
4342     { \int_use:N \c_jCol }
4343   }
4344 }
```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The two others are the elements to draw below and above the diagonal line.

```
4345 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
4346 {
4347   \pgfpicture
4348   \pgf@relevantforpicturesizefalse
4349   \pgfrememberpicturepositiononpagetrue
4350   \@@_qpoint:n { row - #1 }
4351   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4352   \@@_qpoint:n { col - #2 }
4353   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4354   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4355   \@@_qpoint:n { row - \@@_succ:n { #3 } }
4356   \dim_set_eq:NN \l_tmpc_dim \pgf@y
4357   \@@_qpoint:n { col - \@@_succ:n { #4 } }
4358   \dim_set_eq:NN \l_tmpd_dim \pgf@x
4359   \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4360   {
```

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.

```
4361   \CT@arc@
4362   \pgfsetroundcap
4363   \pgfusepathqstroke
4364 }
4365 \pgfset { inner~sep = 1 pt }
4366 \pgfscope
```



```

4367 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4368 \pgfnode { rectangle } { south-west }
4369 { \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }
4370 \endpgfscope
4371 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
4372 \pgfnode { rectangle } { north-east }
4373 { \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
4374 \endpgfpicture
4375 }

```

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. 85.

The command `\CodeAfter` catches everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

4376 \cs_new_protected:Npn \@@_CodeAfter:n #1 \end
4377 {
4378   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
4379   \@@_CodeAfter_i:n
4380 }

```

We catch the argument of the command `\end` (in `#1`).

```

4381 \cs_new_protected:Npn \@@_CodeAfter_i:n #1
4382 {

```

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.

```

4383   \str_if_eq:eeTF \currenenv { #1 }
4384   { \end { #1 } }

```

If this is not the `\end` we are looking for, we put those tokens in `\g_nicematrix_code_after_tl` and we go on searching for the next command `\end` with a recursive call to the command `\@@_CodeAfter:n`.

```

4385   {
4386     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
4387     \@@_CodeAfter:n
4388   }
4389 }

```

We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

4390 \bool_new:N \c_@@_footnotehyper_bool

```

The boolean `\c_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to `true` if the option `footnotehyper` is used.

```

4391 \bool_new:N \c_@@_footnote_bool

```

```

4392 \@@_msg_new:nnn { Unknown~option~for~package }
4393 {
4394   The~option~'\l_keys_key_tl'~is~unknown. \\
4395   If~you~go~on,~it~will~be~ignored. \\
4396   For~a~list~of~the~available~options,~type~H~<return>.
4397 }
4398 {
4399   The~available~options~are~(in~alphabetic~order):~
4400   define-L-C-R,~
4401   footnote,~
4402   footnotehyper,~
4403   renew-dots,~
4404   renew-matrix~and~
4405   transparent.
4406 }
4407 \keys_define:nn { NiceMatrix / Package }
4408 {
4409   define-L-C-R .bool_set:N = \c_@@_define_L_C_R_bool ,
4410   define-L-C-R .default:n = true ,
4411   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
4412   renew-dots .value_forbidden:n = true ,
4413   renew-matrix .code:n = \@@_renew_matrix: ,
4414   renew-matrix .value_forbidden:n = true ,
4415   transparent .meta:n = { renew-dots , renew-matrix } ,
4416   transparent .value_forbidden:n = true,
4417   footnote .bool_set:N = \c_@@_footnote_bool ,
4418   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
4419   unknown .code:n = \@@_error:n { Unknown~option~for~package }
4420 }
4421 \ProcessKeysOptions { NiceMatrix / Package }

4422 \@@_msg_new:nn { footnote~with~footnotehyper~package }
4423 {
4424   You~can't~use~the~option~'footnote'~because~the~package~
4425   footnotehyper~has~already~been~loaded.~
4426   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
4427   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
4428   of~the~package~footnotehyper.\\
4429   If~you~go~on,~the~package~footnote~won't~be~loaded.
4430 }
4431 \@@_msg_new:nn { footnotehyper~with~footnote~package }
4432 {
4433   You~can't~use~the~option~'footnotehyper'~because~the~package~
4434   footnote~has~already~been~loaded.~
4435   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
4436   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
4437   of~the~package~footnote.\\
4438   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
4439 }

4440 \bool_if:NT \c_@@_footnote_bool
4441 {
4442   \@ifclassloaded { beamer }
4443   { \msg_info:nn { nicematrix } { Option~incompatible~with~Beamer } }
4444   {
4445     \@ifpackageloaded { footnotehyper }
4446     { \@@_error:n { footnote~with~footnotehyper~package } }
4447     { \usepackage { footnote } }
4448   }
4449 }

```

```

4450 \bool_if:NT \c_@@_footnotehyper_bool
4451 {
4452   \@ifclassloaded { beamer }
4453   { \@@_info:n { Option-incompatible-with-Beamer } }
4454   {
4455     \@ifpackageloaded { footnote }
4456     { \@@_error:n { footnotehyper-with-footnote-package } }
4457     { \usepackage { footnotehyper } }
4458   }
4459   \bool_set_true:N \c_@@_footnote_bool
4460 }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

Error messages of the package

The following command converts all the elements of a sequence (which are token lists) into strings.

```

4461 \cs_new_protected:Npn \@@_convert_to_str_seq:N #1
4462 {
4463   \seq_clear:N \l_tmpa_seq
4464   \seq_map_inline:Nn #1
4465   {
4466     \seq_put_left:Nx \l_tmpa_seq { \tl_to_str:n { ##1 } }
4467   }
4468   \seq_set_eq:NN #1 \l_tmpa_seq
4469 }

```

The following command creates a sequence of strings (`str`) from a `clist`.

```

4470 \cs_new_protected:Npn \@@_set_seq_of_str_from_clist:Nn #1 #2
4471 {
4472   \seq_set_from_clist:Nn #1 { #2 }
4473   \@@_convert_to_str_seq:N #1
4474 }
4475 \@@_set_seq_of_str_from_clist:Nn \c_@@_types_of_matrix_seq
4476 {
4477   NiceMatrix ,
4478   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
4479 }

```

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`.

```

4480 \cs_new_protected:Npn \@@_error_too_much_cols:
4481 {
4482   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
4483   {
4484     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
4485     { \@@_fatal:n { too-much-cols-for-matrix } }
4486     {
4487       \bool_if:NF \l_@@_last_col_without_value_bool
4488       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
4489     }
4490   }
4491   { \@@_fatal:n { too-much-cols-for-array } }
4492 }

```

The following command must *not* be protected since it's used in an error message.

```

4493 \cs_new:Npn \@@_message_hdotsfor:
4494 {

```

```

4495 \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
4496 { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
4497 }
4498 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
4499 {
4500   You~try~to~use~more~columns~than~allowed~by~your~
4501   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
4502   columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
4503   exterior~columns).~This~error~is~fatal.
4504 }
4505 \@@_msg_new:nn { too-much-cols-for-matrix }
4506 {
4507   You~try~to~use~more~columns~than~allowed~by~your~
4508   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
4509   number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
4510   'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
4511   This~error~is~fatal.
4512 }

```

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`.

```

4513 \@@_msg_new:nn { too-much-cols-for-array }
4514 {
4515   You~try~to~use~more~columns~than~allowed~by~your~
4516   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
4517   \int_use:N \g_@@_static_num_of_col_int\
4518   ~(plus~the~potential~exterior~ones).~
4519   This~error~is~fatal.
4520 }
4521 \@@_msg_new:nn { last-col-not-used }
4522 {
4523   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
4524   in~your~\@@_full_name_env:~However,~you~can~go~on.
4525 }
4526 \@@_msg_new:nn { columns-not-used }
4527 {
4528   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
4529   \g_@@_static_num_of_col_int\
4530   columns~but~you~use~only~\int_use:N \c@jCol.\\
4531   However,~you~can~go~on.
4532 }
4533 \@@_msg_new:nn { in-first-col }
4534 {
4535   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
4536   If~you~go~on,~this~command~will~be~ignored.
4537 }
4538 \@@_msg_new:nn { in-last-col }
4539 {
4540   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
4541   If~you~go~on,~this~command~will~be~ignored.
4542 }
4543 \@@_msg_new:nn { in-first-row }
4544 {
4545   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
4546   If~you~go~on,~this~command~will~be~ignored.
4547 }
4548 \@@_msg_new:nn { in-last-row }
4549 {
4550   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
4551   If~you~go~on,~this~command~will~be~ignored.
4552 }

```

```

4553 \@@_msg_new:nn { bad-option-for-line-style }
4554 {
4555     Since-you-haven't-loaded-Tikz,~the-only~value~you~can~give~to~'line-style'~
4556     is~'standard'.~If-you-go-on,~this-option~will~be~ignored.
4557 }
4558 \@@_msg_new:nn { Unknown-option-for-xdots }
4559 {
4560     As-for~now~there-is-only-three-options-available-here:~'color',~'line-style'~
4561     and~'shorten'~(and-you-try-to-use~'\l_keys_key_tl').~If-you-go-on,~
4562     this-option~will~be~ignored.
4563 }
4564 \@@_msg_new:nn { Unknown-option-for-rowcolors }
4565 {
4566     As-for~now~there-is-only-one-option~available~here:~'respect-blocks'~
4567     (and-you-try-to-use~'\l_keys_key_tl').~If-you-go-on,~
4568     this-option~will~be~ignored.
4569 }
4570 \@@_msg_new:nn { ampersand-in-light-syntax }
4571 {
4572     You-can't-use-an-ampersand~(\token_to_str &)~to-separate-columns~because
4573     ~you-have-used-the-option~'light-syntax'.~This-error-is-fatal.
4574 }
4575 \@@_msg_new:nn { double-backslash-in-light-syntax }
4576 {
4577     You-can't-use~\token_to_str:N ~\~to-separate-rows~because~you~have~used~
4578     the-option~'light-syntax'.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
4579     (set-by-the-option~'end-of-row').~This-error-is-fatal.
4580 }
4581 \@@_msg_new:nn { standard-cline-in-document }
4582 {
4583     The-key~'standard-cline'~is-available-only~in~the~preamble.~\
4584     If-you-go-on~this~command~will~be~ignored.
4585 }
4586 \@@_msg_new:nn { bad-value-for-baseline }
4587 {
4588     The-value-given-to~'baseline'~(\int_use:N \l_tmpa_int)~is-not~
4589     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
4590     \int_use:N \g_@@_row_total_int\ or-equal-to~'t',~'c'~or~'b'.~\
4591     If-you-go-on,~a-value-of~1~will-be-used.
4592 }
4593 \@@_msg_new:nn { empty-environment }
4594 { Your~\@@_full_name_env:~ is-empty.~This-error-is-fatal. }
4595 \@@_msg_new:nn { unknown-cell-for-line-in-code-after }
4596 {
4597     Your-command~\token_to_str:N\line~\{#1\}~\{#2\}~in-the~'code-after'~
4598     can't-be-executed~because~a~cell~doesn't~exist.~\
4599     If-you-go-on~this~command~will~be~ignored.
4600 }
4601 \@@_msg_new:nn { bad-corner }
4602 {
4603     #1-is-an-incorrect-specification-for-a-corner~(in~the~keys~
4604     'except-corners'~and~'hlines-except-corners').~The-available~
4605     values-are:~NW,~SW,~NE~and~SE.~\
4606     If-you-go-on,~this-specification-of-corner~will~be~ignored.
4607 }
4608 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
4609 {
4610     In-the~\@@_full_name_env:,~you-must-use-the-option~
4611     'last-col'~without~value.~\

```

```

4612     However,~you~can~go~on~for~this~time~
4613     (the~value~'\l_keys_value_tl'~will~be~ignored).
4614 }

4615 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
4616 {
4617     In~\NiceMatrixoptions,~you~must~use~the~option~
4618     'last-col'~without~value.\\
4619     However,~you~can~go~on~for~this~time~
4620     (the~value~'\l_keys_value_tl'~will~be~ignored).
4621 }

4622 \@@_msg_new:nn { Block~too~large }
4623 {
4624     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
4625     too~small~for~that~block. \\
4626 }

4627 \@@_msg_new:nn { unknown~column~type }
4628 {
4629     The~column~type~'#1'~in~your~\@@_full_name_env:\
4630     is~unknown. \\
4631     This~error~is~fatal.
4632 }

4633 \@@_msg_new:nn { angle~option~in~NiceTabular }
4634 {
4635     You~should~not~the~option~between~angle~brackets~(<~and~>)~for~a~command~
4636     \token_to_str:N \Block\ in~\{NiceTabular\}.~However,~you~can~go~on.
4637 }

4638 \@@_msg_new:nn { tabularnote~forbidden }
4639 {
4640     You~can't~use~the~command~\token_to_str:N\tabularnote\
4641     ~in~a~\@@_full_name_env:~.~This~command~is~available~only~in~
4642     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
4643     If~you~go~on,~this~command~will~be~ignored.
4644 }

4645 \@@_msg_new:nn { bottomule~without~booktabs }
4646 {
4647     You~can't~use~the~option~'tabular/bottomrule'~because~you~haven't~
4648     loaded~'booktabs'.\\
4649     If~you~go~on,~this~option~will~be~ignored.
4650 }

4651 \@@_msg_new:nn { enumitem~not~loaded }
4652 {
4653     You~can't~use~the~command~\token_to_str:N\tabularnote\
4654     ~because~you~haven't~loaded~'enumitem'.\\
4655     If~you~go~on,~this~command~will~be~ignored.
4656 }

4657 \@@_msg_new:nn { Wrong~last~row }
4658 {
4659     You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
4660     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
4661     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
4662     last~row.~You~can~avoid~this~problem~by~using~'last-row'~
4663     without~value~(more~compilations~might~be~necessary).
4664 }

4665 \@@_msg_new:nn { Yet~in~env }
4666 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }

4667 \@@_msg_new:nn { Outside~math~mode }
4668 {
4669     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
4670     (and~not~in~\token_to_str:N \vcenter).\\

```

```

4671     This-error-is-fatal.
4672 }
4673 \@@_msg_new:nn { Bad-value-for-letter-for-dotted-lines }
4674 {
4675     The-value-of-key~'\l_keys_key_tl'~must-be-of-length~1.\\
4676     If-you-go-on,~it-will-be-ignored.
4677 }
4678 \@@_msg_new:nnn { Unknown-key-for-notes }
4679 {
4680     The-key~'\l_keys_key_tl'~is-unknown.\\
4681     If-you-go-on,~it-will-be-ignored. \\
4682     For-a-list-of-the-available-keys-about-notes,~type-H~<return>.
4683 }
4684 {
4685     The-available-options-are~(in-alphabetic-order):~
4686     bottomrule,~
4687     code-after,~
4688     code-before,~
4689     enumitem-keys,~
4690     enumitem-keys-para,~
4691     para,~
4692     label-in-list,~
4693     label-in-tabular~and~
4694     style.
4695 }
4696 \@@_msg_new:nnn { Unknown-key-for-NiceMatrixOptions }
4697 {
4698     The-key~'\l_keys_key_tl'~is-unknown-for-the-command~
4699     \token_to_str:N \NiceMatrixOptions. \\
4700     If-you-go-on,~it-will-be-ignored. \\
4701     For-a-list-of-the~*principal*~available-keys,~type-H~<return>.
4702 }
4703 {
4704     The-available-options-are~(in-alphabetic-order):~
4705     allow-duplicate-names,~
4706     cell-space-bottom-limit,~
4707     cell-space-top-limit,~
4708     code-for-first-col,~
4709     code-for-first-row,~
4710     code-for-last-col,~
4711     code-for-last-row,~
4712     create-extra-nodes,~
4713     create-medium-nodes,~
4714     create-large-nodes,~
4715     end-of-row,~
4716     first-col,~
4717     first-row,~
4718     hlines,~
4719     hvlines,~
4720     hvlines-except-corners,~
4721     last-col,~
4722     last-row,~
4723     left-margin,~
4724     letter-for-dotted-lines,~
4725     light-syntax,~
4726     notes~(several subkeys),~
4727     nullify-dots,~
4728     renew-dots,~
4729     renew-matrix,~
4730     right-margin,~
4731     small,~
4732     transparent,~
4733     vlines,~

```

```

4734     xdots/color,~
4735     xdots/shorten~and~
4736     xdots/line-style.
4737 }
4738 \@@_msg_new:nnn { Unknown~option~for~NiceArray }
4739 {
4740     The~option~'\l_keys_key_tl'~is~unknown~for~the~environment~
4741     \{NiceArray\}. \\
4742     If~you~go~on,~it~will~be~ignored. \\
4743     For~a~list~of~the~*principal*~available~options,~type~H~<return>.
4744 }
4745 {
4746     The~available~options~are~(in~alphabetic~order):~
4747     b,~
4748     baseline,~
4749     c,~
4750     cell-space-bottom-limit,~
4751     cell-space-top-limit,~
4752     code-after,~
4753     code-for-first-col,~
4754     code-for-first-row,~
4755     code-for-last-col,~
4756     code-for-last-row,~
4757     colortbl-like,~
4758     columns-width,~
4759     create-extra-nodes,~
4760     create-medium-nodes,~
4761     create-large-nodes,~
4762     extra-left-margin,~
4763     extra-right-margin,~
4764     first-col,~
4765     first-row,~
4766     hlines,~
4767     hvlines,~
4768     last-col,~
4769     last-row,~
4770     left-margin,~
4771     light-syntax,~
4772     name,~
4773     notes/bottomrule,~
4774     notes/para,~
4775     nullify-dots,~
4776     renew-dots,~
4777     right-margin,~
4778     rules/color,~
4779     rules/width,~
4780     small,~
4781     t,~
4782     vlines,~
4783     xdots/color,~
4784     xdots/shorten~and~
4785     xdots/line-style.
4786 }

```

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).

```

4787 \@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
4788 {
4789     The~option~'\l_keys_key_tl'~is~unknown~for~the~
4790     \@@_full_name_env:. \\
4791     If~you~go~on,~it~will~be~ignored. \\
4792     For~a~list~of~the~*principal*~available~options,~type~H~<return>.
4793 }

```



```

4794 {
4795   The~available~options~are~(in~alphabetic~order):~
4796   b,~
4797   baseline,~
4798   c,~
4799   cell-space-bottom-limit,~
4800   cell-space-top-limit,~
4801   code-after,~
4802   code-for-first-col,~
4803   code-for-first-row,~
4804   code-for-last-col,~
4805   code-for-last-row,~
4806   colortbl-like,~
4807   columns-width,~
4808   create-extra-nodes,~
4809   create-medium-nodes,~
4810   create-large-nodes,~
4811   extra-left-margin,~
4812   extra-right-margin,~
4813   first-col,~
4814   first-row,~
4815   hlines,~
4816   hvlines,~
4817   l,~
4818   last-col,~
4819   last-row,~
4820   left-margin,~
4821   light-syntax,~
4822   name,~
4823   nullify-dots,~
4824   r,~
4825   renew-dots,~
4826   right-margin,~
4827   rules/color,~
4828   rules/width,~
4829   small,~
4830   t,~
4831   vlines,~
4832   xdots/color,~
4833   xdots/shorten~and~
4834   xdots/line-style.
4835 }

4836 \@_msg_new:nnn { Unknown~option~for~NiceTabular }
4837 {
4838   The~option~'\l_keys_key_tl'~is~unknown~for~the~environment~
4839   \{NiceTabular\}. \\
4840   If~you~go~on,~it~will~be~ignored. \\
4841   For~a~list~of~the~*principal*~available~options,~type~H~<return>.
4842 }
4843 {
4844   The~available~options~are~(in~alphabetic~order):~
4845   b,~
4846   baseline,~
4847   c,~
4848   cell-space-bottom-limit,~
4849   cell-space-top-limit,~
4850   code-after,~
4851   code-for-first-col,~
4852   code-for-first-row,~
4853   code-for-last-col,~
4854   code-for-last-row,~
4855   colortbl-like,~
4856   columns-width,~

```

```

4857     create-extra-nodes,~
4858     create-medium-nodes,~
4859     create-large-nodes,~
4860     extra-left-margin,~
4861     extra-right-margin,~
4862     first-col,~
4863     first-row,~
4864     hlines,~
4865     hvlines,~
4866     last-col,~
4867     last-row,~
4868     left-margin,~
4869     light-syntax,~
4870     name,~
4871     notes/bottomrule,~
4872     notes/para,~
4873     nullify-dots,~
4874     renew-dots,~
4875     right-margin,~
4876     rules/color,~
4877     rules/width,~
4878     t,~
4879     vlines,~
4880     xdots/color,~
4881     xdots/shorten-and~
4882     xdots/line-style.
4883 }

4884 \@@_msg_new:nnn { Duplicate-name }
4885 {
4886     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
4887     the~same~environment~name~twice.~You~can~go~on,~but,~
4888     maybe,~you~will~have~incorrect~results~especially~
4889     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
4890     message~again,~use~the~option~'allow-duplicate-names'~in~
4891     '\token_to_str:N \NiceMatrixOptions'.\\
4892     For~a~list~of~the~names~already~used,~type~H~<return>. \\
4893 }
4894 {
4895     The~names~already~defined~in~this~document~are:~
4896     \seq_use:Nnnn \g_@@_names_seq { ,~ } { ,~ } { ~and~ }.
4897 }

4898 \@@_msg_new:nn { Option-auto-for-columns-width }
4899 {
4900     You~can't~give~the~value~'auto'~to~the~option~'columns-width'~here.~
4901     If~you~go~on,~the~option~will~be~ignored.
4902 }

```

18 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 StackExchange⁴⁵, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁴⁶

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

⁴⁵cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁴⁶Before this version, there was an error when using `nicematrix` with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by `nicematrix` because they use the options `overlay` and `remember picture`.

Changes between version 2.1.3 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`).

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 StackExchange⁴⁷, optimization of Tikz externalization is disabled in the environments of `nicematrix` when the class `standalone` or the package `standalone` is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

Changes between version 3.5 and 3.6

LaTeX counters `iRow` and `jCol` available in the cells of the array.

Addition of `\normalbaselines` before the construction of the array: in environments like `{align}` of `amsmath` the value of `\baselineskip` is changed and if the options `first-row` and `last-row` were used in an environment of `nicematrix`, the position of the delimiters was wrong.

A warning is written in the `.log` file if an obsolete environment is used.

There is no longer artificial errors `Duplicate-name` in the environments of `amsmath`.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.

New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programming for the command `\Block` when the block has only one row. With this programming, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with `\multicolumn`. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands `\NiceMatrixLastEnv` and `\OnlyMainNiceMatrix`.

New options `create-medium-nodes` and `create-large-nodes`.

Changes between version 3.9 and 3.10

New option `light-syntax` (and `end-of-row`).

New option `dotted-lines-margin` for fine tuning of the dotted lines.

⁴⁷cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

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 keyword `\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 `!\qqquad` is used in the preamble of the array.

It's now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.

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

Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).

Environment `{NiceTabular*}`

Command `\Vdotsfor` similar to `\Hdotsfor`

The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.

Key `respect-blocks` for `\rowcolors` (with a *s*) in the `code-before`.

The variable `\g_nicematrix_code_before_tl` is now public.

The key `baseline` can take in as value of the form *line-i* to align the `\hline` in the row *i*.

The key `hvlines-except-corners` may take in as value a list of corner (eg: NW,SE).

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

	Symbols	
@@ commands:		<code>\g_@@_HVdotsfor_lines_tl</code> 1025, 2159, 2969, 3045, 4495
<code>\@@_Block:</code>	1004, 4103	<code>\@@_Hdotsfor:</code> 934, 1001, 2958
<code>\@@_Block_i</code>	4104, 4105	<code>\@@_Hdotsfor:nnnn</code> 2971, 2983
<code>\@@_Block_ii:nnnnn</code>	4105, 4106	<code>\@@_Hdotsfor_i</code> 2961, 2967
<code>\@@_Block_iii:nnnnnn</code>	4133, 4134	<code>\@@_Hline:</code> 999, 3656
<code>\@@_Cdots</code>	929, 994, 2833	<code>\@@_Hline_i:n</code> 3656, 3657, 3663
<code>\g_@@_Cdots_lines_tl</code>	1020, 2163	<code>\@@_Hline_ii:nn</code> 3660, 3663
<code>\@@_Cell:</code>	196, 733, 1381, 1425, 1442, 1963, 2925, 2926, 2927, 2928, 2929, 2930, 2931, 2932, 2933, 2934, 2935, 2936	<code>\@@_Hline_iii:n</code> 3661, 3664
<code>\@@_CodeAfter:n</code>	1008, 4376, 4387	<code>\@@_Hspace:</code> 1000, 2908
<code>\@@_CodeAfter_i:n</code>	4379, 4381	<code>\@@_Iddots</code> 932, 997, 2885
<code>\@@_Ddots</code>	931, 996, 2863	<code>\g_@@_Iddots_lines_tl</code> 1024, 2162
<code>\g_@@_Ddots_lines_tl</code>	1023, 2161	<code>\@@_Ldots</code> 928, 933, 993, 2818
		<code>\g_@@_Ldots_lines_tl</code> 1021, 2164
		<code>\l_@@_Matrix_bool</code> 237, 1222, 1296, 1954

\l_@@_NiceArray_bool	\g_@@_col_total_int
..... 234, 320, 1063, 1174, 1234,	738, 1016, 1215, 1796, 1797, 1828, 1833,
1327, 1339, 1930, 3559, 3560, 3652, 3653, 4309	1838, 1839, 1892, 1998, 2001, 2006, 2013,
\g_@@_NiceMatrixBlock_int	2057, 2501, 3104, 3930, 3940, 3974, 4061, 4159
..... 230, 3894, 3899, 3902, 3913	\l_@@_color_tl
\l_@@_NiceTabular_bool	276, 4129, 4138, 4143
..... 150, 158, 235, 740, 865, 1039,	\@@_colortbl_like:
1118, 1120, 1266, 1270, 1328, 1340, 1981,	936, 1010
1990, 2421, 2429, 3116, 3833, 3840, 4108, 4317	\l_@@_colortbl_like_bool 400, 558, 1010, 1315
\@@_OnlyMainNiceMatrix:n	\c_@@_colortbl_loaded_bool ...
1006, 3446	82, 86, 961
\@@_OnlyMainNiceMatrix_i:n 3449, 3456, 3459	\@@_columncolor
\@@_Vdots	1116, 3231
930, 995, 2848	\@@_columncolor_preamble
\g_@@_Vdots_lines_tl	940, 3437
1022, 2160	\c_@@_columncolor_regex
\@@_Vdotsfor:	205, 1318
1002, 3043	\l_@@_columns_width_dim
\@@_Vdotsfor:nnnn 231, 543, 665, 1765, 1771, 3895, 3901
3047, 3058	\g_@@_com_or_env_str
\@@_W:	247, 250
1300, 1369	\@@_computations_for_large_nodes: ...
\@@_actually_diagbox:nnnnnn 4152, 4329, 4345 4001, 4014, 4019
\@@_actually_draw_Cdots:	\@@_computations_for_medium_nodes: ..
2410, 2414 3921, 3990, 4000, 4011
\@@_actually_draw_Ddots:	\@@_compute_a_corner:nnnnnn
2537, 2541 3705, 3707, 3709, 3711, 3716
\@@_actually_draw_Iddots:	\@@_compute_corners:
2588, 2592	2094, 3697
\@@_actually_draw_Ldots: .	\@@_construct_preamble:n
2368, 2372, 3034	1187, 1293
\@@_actually_draw_Vdots: .	\@@_convert_to_str_seq:N
2464, 2468, 3109	4461, 4473
\@@_adapt_S_column:	\@@_create_col_nodes:
166, 181, 1038	1660, 1688, 1707
\@@_after_array:	\@@_create_large_nodes:
1289, 1994	2080, 3995
\@@_analyze_end:Nn	\@@_create_medium_and_large_nodes: ..
1656, 1701 2077, 4006
\l_@@_argspec_tl	\@@_create_medium_nodes:
2816, 2078, 3985
2817, 2818, 2833, 2848, 2863, 2885, 2965,	\@@_create_nodes: 3992, 4003, 4013, 4016, 4057
2966, 2967, 3041, 3042, 3043, 3139, 3140, 3141	\@@_create_row_node:
\@@_array:	876, 908, 945
860, 1657, 1684	\@@_cut_on_hyphen:w
\l_@@_auto_columns_width_bool	3190, 3212, 3213, 3246, 3247, 3276, 3307, 3318
..... 425, 542, 1764, 1768, 3889	\g_@@_ddots_int
\l_@@_baseline_str	2067, 2561, 2562
414,	\@@_def_env:nnn
415, 535, 536, 537, 538, 873, 1236, 1491, 1936, 1947, 1948, 1949, 1950, 1951
1503, 1508, 1510, 1515, 1520, 1593, 1594, 1595	\@@_define_L_C_R:
\@@_begin_of_NiceMatrix:nn	218, 1186
1952, 1973	\c_@@_define_L_C_R_bool ...
\@@_begin_of_row:	217, 1186, 4409
737, 761, 1850	\@@_define_com:nnn
\l_@@_block_auto_columns_width_bool 4293, 4301, 4302, 4303, 4304, 4305
..... 1052, 1769, 3882, 3887, 3897, 3907	\g_@@_delta_x_one_dim
\g_@@_blocks_seq 272, 1054, 2110, 4118, 4133	2069, 2564, 2574
\c_@@_booktabs_loaded_bool 24, 30, 944, 1562	\g_@@_delta_x_two_dim
\l_@@_cell_box	2071, 2615, 2625
739,	\g_@@_delta_y_one_dim
785, 787, 793, 801, 802, 803, 804, 806,	2070, 2566, 2574
809, 811, 813, 830, 946, 1129, 1131, 1441,	\g_@@_delta_y_two_dim
1449, 1851, 1873, 1876, 1878, 1893, 1914,	2072, 2617, 2625
1918, 3124, 3127, 3131, 4162, 4232, 4237, 4324	\@@_diagbox:nn
\l_@@_cell_space_bottom_limit_dim ...	1009, 4325
..... 403, 469, 804	\@@_dotfill:
\l_@@_cell_space_top_limit_dim 402, 467, 802	4314
\@@_cellcolor	\@@_dotfill_i:
1112, 3267, 3417, 3418	4319, 4321
\@@_cellcolor_tabular	\@@_dotfill_ii:
938, 3423	4318, 4321, 4322
\g_@@_cells_seq	\@@_dotfill_iii:
1695, 1696, 1697, 1699	4322, 4323
\@@_chessboardcolors	\@@_double_int_eval:n
1117, 3410	3135, 3149, 3150
\@@_cline	\g_@@_dp_ante_last_row_dim
129, 992	764, 977
\@@_cline_i:nn 764, 765, 980, 981, 1130, 1131, 1253
130, 131, 143, 146	\g_@@_dp_row_zero_dim
\@@_cline_i:w 784, 785, 971, 972, 1246, 1587, 1613
131, 132	\@@_draw_Cdots:nnn
\l_@@_code_before_bool	2396
..... 261, 532, 559, 880, 1059, 1069,	\@@_draw_Ddots:nnn
1714, 1728, 1746, 1777, 1803, 1826, 2040, 2140	2529
\l_@@_code_before_tl ..	\@@_draw_Iddots:nnn
260, 531, 1060, 1119	2580
\l_@@_code_for_first_col_tl	\@@_draw_Ldots:nnn
481, 1862	2354
\l_@@_code_for_first_row_tl .	\@@_draw_Vdots:nnn
485, 749, 4223	2450
\l_@@_code_for_last_col_tl	\@@_draw_blocks:
483, 1902	2110, 4132
\l_@@_code_for_last_row_tl .	\@@_draw_dotted_lines:
487, 756, 4226	2093, 2148
	\@@_draw_dotted_lines_i:
	2151, 2155
	\@@_draw_hlines:
	2105, 3649

\@@_draw_line:	2394,	2217, 2221, 2230, 2238, 2318, 2348, 2388,
2448, 2527, 2578, 2629, 2631, 3188, 3850, 3880		2486, 2553, 2604, 2988, 3016, 3084, 3094, 3096
\@@_draw_line_ii:nn	3168, 3172	\l_@@_final_j_int
\@@_draw_line_iii:nn	3175, 3179	2084, 2177, 2182, 2189, 2194, 2200, 2210,
\@@_draw_non_standard_dotted_line: ..		2218, 2222, 2231, 2239, 2319, 2349, 2385,
.....	2637, 2639	2426, 2555, 2606, 3009, 3019, 3021, 3063, 3092
\@@_draw_non_standard_dotted_line:n ..		\l_@@_final_open_bool
.....	2642, 2645	2086, 2183,
\@@_draw_standard_dotted_line: ..	2636, 2665	2187, 2190, 2197, 2203, 2207, 2223, 2383,
\@@_draw_standard_dotted_line_i: ..	2730, 2734	2424, 2434, 2445, 2471, 2484, 2492, 2513,
\@@_draw_vlines:	2106, 3556	2551, 2602, 2738, 2753, 2784, 2785, 2986,
\g_@@_empty_cell_bool	268,	3010, 3022, 3061, 3085, 3097, 3158, 3816, 3855
808, 815, 2831, 2846, 2861, 2883, 2905, 2910		\@@_find_extremities_of_line:nnnn ...
\l_@@_empty_corner_cells_seq	2099,
3494, 3500, 3507, 3589, 3595, 3602, 3699, 3772		2171, 2358, 2400, 2455, 2533, 2584
\@@_end_Cell:	198, 797, 1381,	\l_@@_first_col_int
1431, 1446, 1963, 2925, 2926, 2927, 2928,		117, 130,
2929, 2930, 2931, 2932, 2933, 2934, 2935, 2936		279, 280, 477, 713, 737, 1229, 1322, 1710,
\l_@@_end_of_row_tl		1726, 2051, 3203, 3254, 3286, 3315, 3448,
.....	433, 434, 475, 1680, 1681, 4578	3930, 3940, 3974, 4022, 4061, 4275, 4281, 4287
\c_@@_endpgfortikzpicture_tl		\l_@@_first_row_int
.....	39, 43, 2152, 3176, 3819
\c_@@_enumitem_loaded_bool		277, 278, 478, 717, 1014,
.....	25, 33, 297, 586, 591, 602, 607	1244, 1522, 1584, 1597, 1610, 2049, 3923,
\@@_env:		3937, 3964, 4021, 4059, 4181, 4196, 4273, 4589
..	225, 229, 770, 776, 831, 837, 885, 891,	\c_@@_footnote_bool
897, 1083, 1084, 1090, 1091, 1098, 1099,	
1109, 1715, 1718, 1720, 1733, 1739, 1742,		1031, 1291, 4391, 4417, 4440, 4459
1751, 1757, 1760, 1782, 1788, 1791, 1808,		\c_@@_footnotehyper_bool ..
1814, 1820, 1828, 1833, 1839, 2119, 2229,		4390, 4418, 4450
2297, 2336, 2347, 2997, 3015, 3072, 3090,		\@@_full_name_env:
3161, 3163, 3182, 3185, 3727, 3746, 3764,	
3943, 3945, 3953, 4064, 4073, 4091, 4177,		248, 4501, 4508, 4516, 4524,
4184, 4186, 4199, 4201, 4211, 4216, 4217, 4218		4528, 4594, 4610, 4629, 4641, 4660, 4669, 4790
\g_@@_env_int	224, 225, 227, 1051,	\@@_hdottedline:
1057, 1061, 1071, 1075, 1078, 1087, 1088,		998, 3801
1095, 1096, 1139, 1142, 1157, 1160, 2005,		\@@_hdottedline:n
2026, 2044, 2047, 2060, 2136, 3346, 3361, 4100		3809, 3813
\@@_error:n	12, 300, 321,	\@@_hdottedline_i:
445, 454, 612, 653, 664, 673, 678, 696, 703,		3804, 3806
709, 715, 720, 729, 731, 1210, 1220, 1225,		\@@_hdottedline_i:n
1525, 1567, 1600, 3339, 4109, 4419, 4446, 4456		3818, 3822
\@@_error:nn	13, 550,	\@@_hline:nn
2821, 2824, 2836, 2839, 2851, 2854, 2867,		3563, 3654, 3672
2868, 2873, 2874, 2889, 2890, 2895, 2896, 3713		\@@_hline_i:nn
\@@_error:nnn	14, 3166	2103, 3566, 3569
\@@_error_too_much_cols:	1350, 4480	\@@_hline_i_complete:nn
\@@_everycr:	902, 966, 969	2103, 3647
\@@_everycr_i:	902, 903	\@@_hline_ii:nnnn
\l_@@_except_corners_clist		3581, 3609, 3648
.....	421, 518, 522, 3483, 3579, 3700	\l_@@_hlines_bool ..
\l_@@_exterior_arraycolsep_bool		420, 489, 494, 524, 909, 2105
.....	416, 661, 1330, 1342	\g_@@_ht_last_row_dim
\l_@@_extra_left_margin_dim
.....	431, 510, 1190, 1881	766, 978, 979, 1128, 1129, 1252
\l_@@_extra_right_margin_dim		\g_@@_ht_row_one_dim
.....	432, 511, 1202, 1922, 2504	792, 793, 975, 976
\@@_extract_coords_values:	4082, 4089	\g_@@_ht_row_zero_dim
\@@_fatal:n	15, 241,
1042, 1665, 1669, 1671, 1704, 4485, 4488, 4491		786, 787, 973, 974, 1247, 1586, 1612
\@@_fatal:nn	16, 1376	\@@_i:
\l_@@_final_i_int		3923, 3925,
.....	2083, 2176, 2181, 2184, 2209,	3926, 3927, 3928, 3937, 3943, 3945, 3946,
		3947, 3948, 3953, 3954, 3955, 3956, 3964,
		3967, 3969, 3970, 3971, 4023, 4025, 4028,
		4029, 4033, 4034, 4059, 4064, 4066, 4068,
		4072, 4073, 4084, 4091, 4093, 4095, 4099, 4100
		\g_@@_iddots_int
		2068, 2612, 2613
		\l_@@_in_env_bool
		233, 320, 1042, 1043
		\c_@@_in_preamble_bool ..
		21, 22, 23, 582, 598
		\@@_info:n
		4453
		\l_@@_initial_i_int
		2081,
		2174, 2249, 2252, 2277, 2285, 2289, 2298,
		2306, 2316, 2337, 2379, 2436, 2438, 2480,
		2545, 2596, 2987, 2988, 2998, 3066, 3076, 3078
		\l_@@_initial_j_int
	
		2082, 2175, 2250, 2257,
		2262, 2268, 2278, 2286, 2290, 2299, 2307,
		2317, 2338, 2376, 2418, 2494, 2496, 2501,
		2547, 2598, 2991, 3001, 3003, 3062, 3063, 3074
		\l_@@_initial_open_bool
	
		2085, 2251, 2255, 2258, 2265, 2271,
		2275, 2291, 2374, 2416, 2433, 2443, 2471,

2478, 2490, 2543, 2594, 2736, 2783, 2985,	\l_@@_medium_nodes_bool 427, 500, 2074, 4213
2992, 3004, 3060, 3067, 3079, 3157, 3815, 3854	\@@_message_hdotsfor: 4493, 4501, 4508, 4516
\@@_instruction_of_type:nn	\@@_msg_new:nn
..... 841, 2826, 2841, 2856, 2877, 2899 17, 4422, 4431, 4498, 4505, 4513,
\l_@@_inter_dots_dim	4521, 4526, 4533, 4538, 4543, 4548, 4553,
. 404, 405, 2090, 2741, 2748, 2759, 2767,	4558, 4564, 4570, 4575, 4581, 4586, 4593,
2774, 2779, 2791, 2799, 3845, 3848, 3876, 3878	4595, 4601, 4608, 4615, 4622, 4627, 4633,
\g_@@_internal_code_after_tl 255,	4638, 4645, 4651, 4657, 4665, 4667, 4673, 4898
1414, 1465, 2107, 2108, 3671, 3808, 4150, 4327	\@@_msg_new:nnn
\@@_intersect_our_row:nnnn 3399	18, 4392, 4678, 4696, 4738, 4787, 4836, 4884
\@@_intersect_our_row_p:nnnn 3369	\@@_msg_redirect_name:nn 19, 667
\@@_j: 3930, 3932,	\@@_multicolumn:nnn 1003, 2914
3933, 3934, 3935, 3940, 3943, 3945, 3948,	\g_@@_multicolumn_cells_seq
3950, 3951, 3953, 3956, 3958, 3959, 3974, 1012, 2944, 3948, 3956, 4078
3977, 3979, 3980, 3981, 4036, 4038, 4041,	\g_@@_multicolumn_sizes_seq 1013, 2946, 4079
4043, 4047, 4048, 4061, 4064, 4065, 4067,	\g_@@_name_env_str 246,
4072, 4073, 4085, 4091, 4092, 4094, 4099, 4100	251, 252, 1036, 1037, 1703, 1931, 1932,
\l_@@_l_dim	1940, 1941, 1970, 1979, 1987, 2142, 4297, 4482
..... 2714, 2715, 2728, 2729, 2741, 2747,	\l_@@_name_str 426, 552, 772, 775,
2758, 2766, 2774, 2779, 2791, 2792, 2799, 2800	833, 836, 893, 896, 1137, 1146, 1149, 1155,
\l_@@_large_nodes_bool 428, 501, 2076, 2080	1164, 1167, 1719, 1720, 1741, 1742, 1759,
\g_@@_last_col_found_bool 287,	1760, 1790, 1791, 1816, 1819, 1835, 1838,
1019, 1216, 1284, 1795, 1824, 1890, 1997, 2054	2008, 2012, 2029, 2033, 4069, 4072, 4096, 4099
\l_@@_last_col_int	\g_@@_names_seq 232, 549, 551, 4896
.. 285, 286, 654, 689, 691, 704, 716, 730,	\l_@@_nb_cols_int
1081, 1153, 1159, 1166, 1219, 1334, 1959, 4264, 4269, 4272, 4276, 4282, 4288
1961, 1998, 2001, 2053, 2460, 2499, 2823,	\l_@@_nb_rows_int 4263, 4268, 4279
2838, 2874, 2896, 4277, 4283, 4289, 4484, 4502	\@@_newcolumnntype 919, 1299, 1300
\l_@@_last_col_without_value_bool ...	\@@_node_for_multicolumn:nn 4080, 4087
..... 284, 688, 1999, 4487	\@@_node_for_the_cell: 812, 817, 1877, 1923
\l_@@_last_empty_column_int	\@@_node_position: .. 1090, 1092, 1098, 1100
..... 3737, 3738, 3751, 3757, 3770	\@@_not_in_exterior:nnnn 3391
\l_@@_last_empty_row_int	\@@_not_in_exterior_p:nnnn 3363
..... 3719, 3720, 3733, 3754	\l_@@_notes_above_space_dim 422, 423
\l_@@_last_row_int .. 281, 282, 479, 754,	\l_@@_notes_bottomrule_bool
913, 1079, 1124, 1134, 1141, 1148, 1204, 570, 707, 726, 1560
1208, 1211, 1228, 1250, 1682, 1683, 1858,	\l_@@_notes_code_after_tl 568, 1569
1859, 1899, 1900, 2020, 2363, 2405, 2853,	\l_@@_notes_code_before_tl 566, 1545
2868, 2890, 3125, 3454, 3462, 4225, 4285, 4659	\@@_notes_label_in_list:n 293, 308, 316, 578
\l_@@_last_row_without_value_bool ...	\@@_notes_label_in_tabular:n . 292, 329, 575
..... 283, 1136, 1206, 2018	\l_@@_notes_para_bool .. 564, 705, 724, 1546
\l_@@_left_delim_dim	\@@_notes_style:n
..... 1172, 1176, 1181, 1648, 1879 291, 294, 308, 316, 332, 337, 572
\l_@@_left_delim_tl 1033, 3844	\l_@@_nullify_dots_bool
\l_@@_left_margin_dim 424, 499, 2830, 2845, 2860, 2882, 2904
..... 429, 504, 1189, 1880, 3834, 4052	\l_@@_number_of_notes_int 290, 323, 333, 343
\l_@@_letter_for_dotted_lines_str ...	\@@_old_CT@arc@ 1044, 2144
..... 672, 680, 681, 1374	\@@_old_arraycolsep_dim 269
\l_@@_light_syntax_bool	\@@_old_cdots 986, 2845
..... 413, 473, 1192, 1197, 2021	\@@_old_ddots 988, 2882
\@@_light_syntax_i 1673, 1676	\@@_old_dotfill 4313, 4316, 4324
\@@_line 2123, 3141	\@@_old_dotfill: 1007
\@@_line_i:nn 3148, 3155	\l_@@_old_iRow_int 256, 948, 2168
\@@_line_with_light_syntax:n ... 1687, 1691	\@@_old_ialign: 875, 982, 2109
\@@_line_with_light_syntax_i:n	\@@_old_iddots 989, 2904
..... 1686, 1692, 1693	\l_@@_old_jCol_int 257, 951, 2169
\@@_math_toggle_token: 149,	\@@_old_ldots 985, 2830
799, 1852, 1869, 1894, 1910, 4122, 4369, 4373	\@@_old_multicolumn 2913, 2920
\g_@@_max_cell_width_dim	\@@_old_pgfulil@check@rerun 75, 79
..... 805, 806, 1053, 1770, 3888, 3914	\@@_old_vdots 987, 2860
\l_@@_max_delimiter_width_bool	\l_@@_parallelize_diags_bool
..... 436, 472, 1280 417, 418, 496, 2065, 2559, 2610
\c_@@_max_l_dim 2728, 2733	

\@@_patch_preamble:n	\l_@@_right_delim_dim
..... 1312, 1354, 1388, 1417, 1467, 1479 1173, 1177, 1183, 1651, 1920
\@@_patch_preamble_i:n 1358, 1359, 1360, 1379	\l_@@_right_delim_tl
\@@_patch_preamble_ii:nn 1034, 3847
..... 1361, 1362, 1363, 1385	\l_@@_right_margin_dim
\@@_patch_preamble_iii:n . 1364, 1390, 1398 430, 506, 1201, 1921, 2503, 3841, 4055
\@@_patch_preamble_iii_i:n 1393, 1395	\@@_rotate:
\@@_patch_preamble_iv:nnn 1005, 3114
..... 1365, 1366, 1367, 1420	\@@_rotate_i:
\@@_patch_preamble_ix:n 3118, 3120
..... 1472, 1482	\@@_rotate_ii:
\@@_patch_preamble_v:nnnn 1368, 1369, 1436 3117, 3120, 3121
\@@_patch_preamble_vi:n	\@@_rotate_iii:
..... 1370, 1455 3121, 3122
\@@_patch_preamble_vii:n	\g_@@_row_of_col_done_bool
..... 1375, 1461 259, 906, 1035, 1725
\@@_patch_preamble_viii:n	\g_@@_row_total_int
..... 1383, 1434, 1453, 1459, 1469, 1485 1015, 1227,
\@@_pgf_rect_node:nnn	1523, 1598, 2020, 2027, 2034, 2050, 3029,
..... 376, 4215	3923, 3937, 3964, 4059, 4158, 4181, 4196, 4590
\@@_pgf_rect_node:nnnn	\@@_rowcolor 1114, 3195, 3352, 3353, 3374, 3379
..... 351, 4063, 4090, 4176, 4210	\@@_rowcolor_tabular
\c_@@_pgfortikzpicture_tl 939, 3428
..... 38, 42, 2150, 3174, 3817	\@@_rowcolors
\@@_picture_position: 1084, 1092, 1100 1115, 3341
\g_@@_pos_of_blocks_seq 273, 1055,	\@@_rowcolors_i:nnnn
2061, 2097, 2947, 3479, 3575, 3784, 4117, 4337 3347, 3358
\g_@@_pos_of_xdots_seq	\@@_rowcolors_ii:nnnn
..... 274, 1056, 2098, 2314, 3481, 3577 3371, 3386
\@@_pre_array:	\g_@@_rows_seq . 1679, 1681, 1683, 1685, 1687
..... 942, 1171	\l_@@_rules_color_tl .. 258, 458, 1067, 1068
\c_@@_preamble_first_col_tl 1323, 1846	\@@_set_CT@arc@:
\c_@@_preamble_last_col_tl 1335, 1886 151, 1068
\g_@@_preamble_tl	\@@_set_CT@arc@_i:
..... 1297, 1307, 1310, 1320, 1323, 152, 153
1332, 1335, 1344, 1349, 1381, 1387, 1400,	\@@_set_CT@arc@_ii:
1422, 1438, 1457, 1463, 1476, 1484, 1657, 1684 152, 155
\@@_pred:n	\@@_set_final_coords:
..... 118, 148, 1961, 3495, 3508, 3590, 3603 2327, 2352
\@@_provide_pgfsyspdfmark: . 206, 215, 1030	\@@_set_final_coords_from_anchor:n ..
\@@_put_box_in_flow: 1282, 1487, 1650	... 2343, 2391, 2431, 2474, 2489, 2558, 2609
\@@_put_box_in_flow_bis:nn 1281, 1617	\@@_set_initial_coords:
\@@_put_box_in_flow_i: 2322, 2341
..... 1493, 1495	\@@_set_initial_coords_from_anchor:n .
\@@_qpoint:n 2332, 2382, 2423, 2473, 2483, 2550, 2601
..... 228, 1498, 1500,	\@@_set_seq_of_str_from_clist:Nn 4470, 4475
1512, 1528, 1578, 1580, 1604, 1606, 2376,	\@@_set_size:n
2379, 2385, 2388, 2418, 2426, 2436, 2438, 4261, 4270
2480, 2486, 2494, 2496, 2545, 2547, 2553,	\c_@@_siunitx_loaded_bool 159, 163, 168, 186
2555, 2596, 2598, 2604, 2606, 3182, 3185,	\l_@@_small_bool
3202, 3206, 3219, 3221, 3238, 3240, 3253, 655,
3257, 3277, 3283, 3285, 3289, 3312, 3314,	694, 700, 718, 743, 954, 1853, 1895, 2087
3323, 3325, 3518, 3520, 3522, 3613, 3615,	\@@_standard_cline
3617, 3825, 3829, 3836, 3872, 3875, 3877, 114, 991
3969, 3979, 4166, 4168, 4170, 4172, 4192,	\@@_standard_cline:w
4207, 4228, 4229, 4235, 4350, 4352, 4355, 4357 114, 115
\l_@@_radius_dim	\l_@@_standard_cline_bool ... 401, 465, 990
..... 408, 409, 1464,	\c_@@_standard_tl 411, 412, 2635, 3849, 3879
2089, 2392, 2393, 2808, 3803, 3827, 3873, 3874	\g_@@_static_num_of_col_int
\l_@@_real_left_delim_dim 1619, 1634, 1649 275, 1224, 1313, 4517, 4529
\l_@@_real_right_delim_dim 1620, 1646, 1652	\l_@@_stop_loop_bool
\@@_rectanglecolor 2178, 2179,
..... 1113, 3300	2211, 2224, 2233, 2246, 2247, 2279, 2292, 2301
\@@_renew_NC@rewrite@S: 187, 189, 1018	\@@_succ:n 143, 147, 885, 891, 1415,
\@@_renew_dots:	1500, 1808, 1814, 1819, 1820, 1828, 1833,
..... 926, 1011	1838, 1839, 2055, 2385, 2426, 2438, 2486,
\l_@@_renew_dots_bool	2496, 2553, 2555, 2598, 2604, 3206, 3219,
..... 497, 1011, 4411	3240, 3257, 3283, 3289, 3323, 3325, 3395,
\@@_renew_matrix:	3491, 3522, 3560, 3586, 3617, 3653, 3672,
..... 657, 4243, 4413	3789, 3791, 3793, 3795, 3836, 3877, 4029,
\l_@@_respect_blocks_bool	4033, 4043, 4047, 4170, 4172, 4207, 4355, 4357
..... 3337, 3345	\l_@@_suffix_tl
\@@_restore_iRow_jCol: 3991, 4002,
..... 2143, 2166	4012, 4015, 4064, 4072, 4073, 4091, 4099, 4100
\@@_revtex_array:	\@@_tab_or_array_colsep: .. 157, 2378, 2387
..... 852, 863	\c_@@_table_collect_begin_tl . 176, 178, 196
\c_@@_revtex_bool	\c_@@_table_print_tl
..... 46, 48, 51, 862 179, 180, 198
	\l_@@_tabular_width_dim
 236, 868, 870, 1346, 1988
	\g_@@_tabularnotes_seq
 289, 324, 1549, 1555, 1571
	\@@_test_if_cell_in_a_block:nn
 3723, 3741, 3759, 3779

<code>\bool_if:nTF</code>	186, 297, 320, 1216, 2074, 3159, 3401
<code>\bool_lazy_all:nTF</code>	1325, 1337, 2095, 3677, 3688
<code>\bool_lazy_and:nnTF</code>	1767, 1854, 2052, 2432, 2667, 3278, 3308, 3344, 3524, 3619
<code>\bool_lazy_or:nnTF</code>	441, 1521, 1596, 1898, 2471, 2727, 3393, 3724, 3742, 3760, 4157
<code>\bool_lazy_or_p:nn</code>	1857
<code>\bool_not_p:n</code>	1328, 1329, 1330, 1340, 1341, 1342, 1769, 2054
<code>\bool_set:Nn</code>	2475
<code>\g_tmpa_bool</code>	3478, 3485, 3496, 3504, 3509, 3574, 3580, 3591, 3599, 3604, 3684, 3695
<code>\l_tmpb_bool</code> ...	3729, 3743, 3761, 3783, 3796
box commands:	
<code>\box_clear_new:N</code>	946, 1185
<code>\box_dp:N</code>	765, 785, 804, 972, 981, 1131, 1490, 1628, 1641
<code>\box_ht:N</code>	766, 787, 793, 802, 974, 976, 979, 1129, 1489, 1628, 1641, 3130
<code>\box_move_up:nn</code> .	58, 60, 62, 1533, 1589, 1615
<code>\box_rotate:Nn</code>	3124
<code>\box_set_dp:Nn</code>	803, 1490
<code>\box_set_ht:Nn</code>	801, 1489
<code>\box_use:N</code>	344, 1428, 1431, 3131
<code>\box_use_drop:N</code>	809, 813, 830, 1449, 1492, 1533, 1534, 1539, 1542, 1878, 4232, 4237
<code>\box_wd:N</code>	345, 806, 811, 1181, 1183, 1541, 1635, 1647, 1873, 1876, 1914, 1918, 4324
<code>\l_tmpa_box</code>	327, 344, 345, 1180, 1181, 1182, 1183, 1256, 1489, 1490, 1492, 1533, 1534, 1628, 1641
<code>\l_tmpb_box</code>	1621, 1635, 1636, 1647
C	
<code>\c</code>	205, 1319
<code>\Cdots</code>	994, 2836, 2839
<code>\cdots</code>	929, 986
<code>\cellcolor</code>	938, 1112, 3426
<code>\chessboardcolors</code>	1117
<code>\cline</code>	142, 991, 992
clist commands:	
<code>\clist_if_empty:NTF</code>	3483, 3579
<code>\clist_map_inline:Nn</code>	3700
<code>\clist_map_inline:nn</code> .	1966, 3208, 3242, 3274
<code>\clist_new:N</code>	421
<code>\clist_set:Nn</code>	522
<code>\CodeAfter</code>	1008, 1673, 1676, 2124
<code>\color</code>	93, 100, 154, 156, 2361, 2364, 2367, 2403, 2406, 2409, 2452, 2458, 2461, 2536, 2587, 3027, 3030, 3033, 3102, 3105, 3108, 3145, 3201, 3237, 3273, 3306
<code>\colorlet</code>	244, 245, 750, 757, 1863, 1903
<code>\columncolor</code>	940, 1116, 2131, 3442
<code>\cr</code>	126, 144, 1844
<code>\crcr</code>	1709
cs commands:	
<code>\cs_generate_variant:Nn</code> ...	146, 3919, 3920
<code>\cs_gset:Npn</code>	93, 100, 2005, 2012, 2026, 2033, 3912
<code>\cs_gset_eq:NN</code>	181, 215, 965, 1044, 2144
<code>\cs_if_exist:NTF</code>	947, 950, 1045, 1048, 1139, 1146, 1157, 1164, 2168, 2169, 2214, 2227, 2282, 2295, 2995, 3013, 3070, 3088, 3899, 3942, 4183, 4198
<code>\cs_if_exist_p:N</code>	442, 3346, 3726, 3745, 3763
<code>\cs_if_free:NTF</code>	211, 2356, 2398, 2453, 2531, 2582
<code>\cs_if_free_p:N</code>	3161, 3163
<code>\cs_new_protected:Npx</code>	2148, 3172, 3813
<code>\cs_set:Nn</code>	572, 575, 578
<code>\cs_set:Npn</code>	89, 90, 96, 97, 102, 114, 115, 129, 131, 132, 154, 156, 294, 857, 869, 921, 956, 959, 2173, 2235, 2303, 3037, 3112, 3656, 3657, 3663, 3664, 4084, 4085
<code>\cs_set:Npx</code>	870
<code>\cs_set_protected:Npn</code>	4295
D	
<code>\Ddots</code>	996, 2867, 2868, 2873, 2874
<code>\ddots</code>	931, 988
<code>\diagbox</code>	1009, 4148
dim commands:	
<code>\dim_add:Nn</code>	4053
<code>\dim_compare:nNnTF</code>	92, 99, 1346, 1765, 1918, 2516, 3966, 3976, 4190, 4205, 4324
<code>\dim_max:nn</code>	3955, 3959
<code>\dim_min:nn</code>	3947, 3951
<code>\dim_ratio:nn</code>	2574, 2625, 2741, 2746, 2757, 2765, 2774, 2779, 2790, 2798
<code>\dim_set:Nn</code>	3928, 3935, 3946, 3950, 3954, 3958, 3970, 3971, 3980, 3981, 4025, 4038
<code>\dim_set_eq:NN</code>	3926, 3933, 4033, 4047
<code>\dim_sub:Nn</code>	4050
<code>\dim_use:N</code>	3967, 3977, 4028, 4029, 4041, 4042, 4065, 4066, 4067, 4068, 4092, 4093, 4094, 4095
<code>\dim_zero_new:N</code>	3925, 3927, 3932, 3934
<code>\c_max_dim</code>	3926, 3928, 3933, 3935, 3967, 3977, 4180, 4190, 4195, 4205
<code>\l_tmpc_dim</code>	266, 3204, 3205, 3224, 3255, 3256, 3260, 3287, 3288, 3292, 3316, 3317, 3328, 3523, 3531, 3536, 3541, 3547, 3618, 3629, 3634, 3640, 4171, 4178, 4212, 4356, 4359, 4367
<code>\l_tmpd_dim</code>	267, 3222, 3224, 3258, 3261, 3290, 3293, 3326, 3329, 3532, 3536, 3625, 3629, 4173, 4178, 4195, 4202, 4205, 4208, 4212, 4358, 4359, 4371
<code>\dotfill</code>	1007, 4313
<code>\dots</code>	933
<code>\doublerulesep</code>	1409, 3533, 3545, 3626, 3638, 3669
<code>\doublerulesepcolor</code>	96
<code>\draw</code>	2647
E	
<code>\egroup</code>	1290
else commands:	
<code>\else:</code>	240
<code>\endarray</code>	1661, 1689
<code>\endBNiceMatrix</code>	4259
<code>\endbNiceMatrix</code>	4256
<code>\endNiceArray</code>	1984, 1993
<code>\endNiceArrayWithDelims</code>	1935, 1945
<code>\endpgfscope</code>	2709, 4370
<code>\endpNiceMatrix</code>	4247
<code>\endsavenotes</code>	1291

<code>\endtabularnotes</code>	1556
<code>\endVNiceMatrix</code>	4253
<code>\endvNiceMatrix</code>	4250
<code>\everycr</code>	125, 144, 969
exp commands:	
<code>\exp_after:wN</code>	193, 1068, 1312
<code>\exp_args:Ne</code>	2923
<code>\exp_args:NNc</code>	3901
<code>\exp_args:NNe</code>	2919
<code>\exp_args:Nne</code>	1973
<code>\exp_args:NNV</code>	1681, 2818, 2833, 2848, 2863, 2885, 2967, 3043, 3141
<code>\exp_args:NNv</code>	1060
<code>\exp_args:No</code>	2642
<code>\exp_args:NV</code>	1657, 1684, 1686
<code>\exp_args:Nx</code>	4175
<code>\exp_not:N</code>	38, 39, 42, 43, 1402, 3432, 3442, 3815, 3816, 4142
<code>\exp_not:n</code>	849, 2977, 3053, 3434, 4122, 4154, 4334, 4335
<code>\ExplSyntaxOff</code>	213, 2016, 2037, 2063, 2139, 3916
<code>\ExplSyntaxOn</code>	210, 2002, 2023, 2042, 2132, 3909
F	
<code>\fi</code>	104, 1303, 3656, 3673
fi commands:	
<code>\fi:</code>	242
<code>\fontdimen</code>	1529
fp commands:	
<code>\fp_eval:n</code>	2680
<code>\fp_to_dim:n</code>	2717
<code>\futurelet</code>	109
G	
group commands:	
<code>\group_insert_after:N</code>	3117, 3118, 3120, 3121, 4318, 4319, 4321, 4322
H	
<code>\halign</code>	983
<code>\hbox</code>	878, 1264, 1730, 1748, 1775, 1779, 1805
hbox commands:	
<code>\hbox:n</code>	58, 60, 63
<code>\hbox_overlap_left:n</code>	1874
<code>\hbox_overlap_right:n</code>	344, 1916
<code>\hbox_set:Nn</code>	327, 1180, 1182, 1256, 1621, 1636, 4162
<code>\hbox_set:Nw</code>	739, 1188, 1441, 1851, 1893
<code>\hbox_set_end:</code>	800, 1203, 1448, 1870, 1911
<code>\hbox_to_wd:nn</code>	369, 394
<code>\Hdotsfor</code>	1001, 4496
<code>\hdotsfor</code>	934
<code>\hdottedline</code>	998
<code>\heavyrulewidth</code>	1565
<code>\hfil</code>	1369
<code>\hfill</code>	122, 139
<code>\Hline</code>	999, 3659
<code>\hline</code>	102
<code>\hrule</code>	106, 122, 139, 914, 1565
<code>\hskip</code>	105
<code>\Hspace</code>	1000
<code>\hspace</code>	2911
<code>\hss</code>	1369

I	
<code>\ialign</code>	875, 959, 982, 2109
<code>\Iddots</code>	997, 2889, 2890, 2895, 2896
<code>\iddots</code>	53, 932, 989
if commands:	
<code>\if_mode_math:</code>	240
<code>\ifnum</code>	104, 3656, 3673
<code>\ifstandalone</code>	1048
int commands:	
<code>\int_case:nnTF</code>	2865, 2871, 2887, 2893
<code>\int_compare:nNnTF</code>	117, 118, 134, 736, 737, 747, 754, 790, 911, 913, 1079, 1081, 1124, 1134, 1153, 1204, 1208, 1219, 1224, 1228, 1229, 1584, 1610, 1682, 1710, 1896, 2189, 2196, 2200, 2202, 2257, 2264, 2268, 2270, 2363, 2405, 2460, 2499, 2501, 2942, 3029, 3104, 3125, 3217, 3251, 3319, 3321, 3388, 3439, 3453, 3454, 3461, 3462, 3466, 3789, 3791, 3793, 3795, 4225, 4273, 4275, 4277, 4281, 4283, 4285, 4287, 4289
<code>\int_compare_p:n</code>	3279, 3280, 3309, 3310, 3403, 3405
<code>\int_do_until:nNnn</code>	3366
<code>\int_gadd:Nn</code>	2955
<code>\int_gincr:N</code>	735, 763, 1051, 1382, 1433, 1452, 1458, 1801, 1891, 2561, 2612, 3894
<code>\int_if_even:nTF</code>	3416
<code>\int_incr:N</code>	323, 1392
<code>\int_step_inline:nn</code>	3412, 3414
<code>\int_step_inline:nnnn</code>	3721, 3739, 3754, 3757
<code>\c_zero_int</code>	3394
iow commands:	
<code>\iow_now:Nn</code>	70, 208, 2042, 2043, 2045, 2063, 2132, 2133, 2139
<code>\iow_shipout:Nn</code>	2002, 2003, 2010, 2016, 2023, 2024, 2031, 2037, 3909, 3910, 3916
<code>\item</code>	1549, 1555
K	
<code>\kern</code>	63
keys commands:	
<code>\keys_define:nn</code>	437, 456, 463, 516, 562, 614, 650, 684, 698, 711, 722, 3335, 3883, 4125, 4407
<code>\l_keys_key_tl</code>	4394, 4561, 4567, 4675, 4680, 4698, 4740, 4789, 4838
<code>\keys_set:nn</code>	471, 676, 683, 1064, 1065, 1972, 1980, 1989, 2366, 2408, 2463, 2535, 2586, 3032, 3107, 3144, 3343, 3896, 4137
<code>\l_keys_value_tl</code>	4613, 4620, 4886
L	
<code>\Ldots</code>	993, 2821, 2824
<code>\ldots</code>	928, 985
<code>\leaders</code>	122, 139
<code>\left</code>	1259, 1624, 1639
legacy commands:	
<code>\legacy_if:nTF</code>	546
<code>\line</code>	2123, 4597
M	
<code>\makebox</code>	1449
<code>\mathinner</code>	55
mode commands:	
<code>\mode_leave_vertical:</code>	1040, 1427

seq commands:

- `\seq_clear:N` 4463
- `\seq_clear_new:N` 2044, 3699
- `\seq_count:N` 1683
- `\seq_gclear:N` 1054, 1055, 1056, 1571
- `\seq_gclear_new:N` ... 1012, 1013, 1679, 1695
- `\seq_gpop_left:Nn` 1685, 1697
- `\seq_gput_left:Nn` 551, 2944, 2946, 4117, 4118
- `\seq_gput_right:Nn` ... 324, 2314, 2947, 4337
- `\seq_gset_from_clist:Nn` 2047, 2059
- `\seq_gset_split:Nnn` 1681, 1696
- `\seq_if_empty:NTF` 2110
- `\seq_if_empty_p:N` 2097, 2098, 2099
- `\seq_if_exist:NTF` 1071
- `\seq_if_in:NnTF` 549, 3493, 3499, 3506, 3588, 3594, 3601, 3948, 3956, 4482
- `\seq_item:Nn` 1075, 1078, 1087, 1088, 1095, 1096
- `\seq_map_function:NN` 1687
- `\seq_map_inline:Nn` ... 1549, 1555, 1699, 3371, 3479, 3481, 3575, 3577, 3784, 4133, 4464
- `\seq_mapthread_function:NNN` 4077
- `\seq_new:N` 232, 272, 273, 274, 289
- `\seq_put_left:Nn` 4466
- `\seq_put_right:Nn` 3771
- `\seq_set_eq:NN` 3360, 4468
- `\seq_set_filter:NNn` 3362, 3368
- `\seq_set_from_clist:Nn` 4472
- `\seq_use:Nnnn` 2061, 4896
- `\l_tmpa_seq` 3362, 3368, 4463, 4466, 4468
- `\l_tmpb_seq` 3360, 3362, 3368, 3371
- `\setlist` 304, 311, 587, 592, 603, 608

skip commands:

- `\skip_gadd:Nn` 1772
- `\skip_gset:Nn` 1763
- `\skip_gset_eq:NN` 1770, 1771
- `\skip_horizontal:N` 123, 140, 1189, 1190, 1201, 1202, 1231, 1232, 1267, 1268, 1271, 1272, 1286, 1287, 1308, 1464, 1477, 1648, 1649, 1651, 1652, 1713, 1732, 1734, 1750, 1752, 1774, 1781, 1783, 1802, 1807, 1809, 1830, 1842, 1879, 1880, 1881, 1883, 1915, 1920, 1921, 1922
- `\skip_horizontal:n` 345, 1404
- `\skip_vertical:N` 127, 884, 886, 1262, 1263, 1274, 1275, 1543, 1564, 3803
- `\skip_vertical:n` 3130, 3666
- `\g_tmpa_skip` 1763, 1770, 1771, 1772, 1774, 1802
- `\c_zero_skip` 970

`\space` 251, 252

`\stepcounter` 331, 336

str commands:

- `\c_backslash_str` 251
- `\c_colon_str` 681
- `\str_case:nn` 2923
- `\str_case:nnTF` 1236, 1356, 1515, 3702
- `\str_count:N` 1510
- `\str_gclear:N` 2142
- `\str_gset:Nn` 1037, 1932, 1941, 1970, 1979, 1987, 4297
- `\str_if_empty:NTF` 772, 833, 893, 1036, 1137, 1155, 1719, 1741, 1759, 1790, 1816, 1835, 1931, 1940, 2008, 2029, 4069, 4096

- `\str_if_eq:nnTF` 78, 250, 873, 1374, 1397, 1471, 1491, 1593, 1703, 4383
- `\str_if_eq_p:nn` 443
- `\str_if_in:NnTF` 1503
- `\str_new:N` 246, 414, 426, 680
- `\str_range:Nnn` 1507
- `\str_set:Nn` 548, 672
- `\str_set_eq:NN` 552, 681
- `\l_tmpa_str` 548, 549, 551, 552
- `\strut` 1549, 1555

T

- `\tabcolsep` 158, 866, 1267, 1271, 2421, 2429, 3833, 3840
- `\tabskip` 970
- `\tabularnote` 299, 318, 325, 4640, 4653
- `\tabularnotes` 1554

TeX and L^AT_EX 2_ε commands:

- `\@BTnormal` 945
- `\@acol` 856
- `\@acoll` 854
- `\@acolr` 855
- `\@array@array` 858
- `\@arrayacol` 854, 855, 856
- `\@arstrutbox` 765, 766, 972, 974, 976, 979, 981, 1428, 1431, 3130
- `\@currenvir` 4383
- `\@gobblethree` 212
- `\@halignto` 857, 869, 870
- `\@height` 107, 122, 139
- `\@ifclassloaded` 47, 50, 4442, 4452
- `\@ifnextchar` 1017
- `\@ifpackageloaded` 29, 32, 35, 69, 85, 162, 4445, 4455
- `\@mainaux` 70, 208, 2002, 2003, 2010, 2016, 2023, 2024, 2031, 2037, 2042, 2043, 2045, 2063, 2132, 2133, 2139, 3909, 3910, 3916
- `\@tabarray` 871
- `\@tempswafalse` 1303
- `\@tempswatru` 1302
- `\@temptokena` .. 171, 174, 193, 195, 1301, 1312
- `\@whiles` 1303
- `\@width` 107
- `\@xhline` 110
- `\bBigg@` 1180, 1182
- `\c@MaxMatrixCols` 1960, 4510
- `\c@tabularnote` 1538, 1572
- `\col@sep` 866, 867, 1713, 1772, 1830, 1842, 1883, 1915
- `\CT@arc` 89, 90
- `\CT@arc@` 88, 93, 108, 121, 138, 154, 156, 1044, 1565, 2144, 3549, 3642, 3870, 4361
- `\CT@dr` 96, 97
- `\CT@drsc@` ... 95, 100, 3526, 3529, 3621, 3624
- `\CT@everycr` 963
- `\CT@row@color` 965
- `\if@temp` 1303
- `\NC@` 921
- `\NC@find` 172, 200
- `\NC@list` 1303
- `\NC@rewrite@S` 173, 191
- `\new@ifnextchar` 1017
- `\newcol@` 923, 924
- `\nicematrix@redefine@check@rerun` .. 70, 73

4	The blocks	4
5	The rules	5
5.1	Some differences with the classical environments	5
5.1.1	The vertical rules	5
5.1.2	The command <code>\cline</code>	6
5.2	The thickness and the color of the rules	6
5.3	The keys <code>hlines</code> and <code>vlines</code>	6
5.4	The key <code>hvlines</code>	7
5.5	The key <code>hvlines-except-corners</code>	7
5.6	The command <code>\diagbox</code>	8
5.7	Dotted rules	8
6	The color of the rows and columns	9
6.1	Use of <code>colortbl</code>	9
6.2	The tools of <code>nicematrix</code> in the code-before	9
6.3	Color tools with the syntax of <code>colortbl</code>	11
7	The width of the columns	13
8	The exterior rows and columns	14
9	The continuous dotted lines	15
9.1	The option <code>nullify-dots</code>	17
9.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	17
9.3	How to generate the continuous dotted lines transparently	18
9.4	The labels of the dotted lines	19
9.5	Customization of the dotted lines	19
9.6	The dotted lines and the rules	20
10	The code-after	20
11	The notes in the tabulars	21
11.1	The footnotes	21
11.2	The notes of tabular	21
11.3	Customisation of the tabular notes	22
11.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	24
12	Other features	25
12.1	Use of the column type <code>S</code> of <code>siunitx</code>	25
12.2	Alignment option in <code>{NiceMatrix}</code>	25
12.3	The command <code>\rotate</code>	25
12.4	The option <code>small</code>	26
12.5	The counters <code>iRow</code> and <code>jCol</code>	26
12.6	The option <code>light-syntax</code>	27
12.7	The environment <code>{NiceArrayWithDelims}</code>	27
13	Use of Tikz with <code>nicematrix</code>	27
13.1	The nodes corresponding to the contents of the cells	27
13.2	The “medium nodes” and the “large nodes”	28
13.3	The “row-nodes” and the “col-nodes”	30
14	API for the developpers	30
15	Technical remarks	31
15.1	Definition of new column types	31
15.2	Diagonal lines	32
15.3	The “empty” cells	32
15.4	The option <code>exterior-arraycolsep</code>	33
15.5	Incompatibilities	33

16	Examples	33
16.1	Notes in the tabulars	33
16.2	Dotted lines	34
16.3	Dotted lines which are no longer dotted	36
16.4	Width of the columns	37
16.5	How to highlight cells of the matrix	37
16.6	Direct use of the Tikz nodes	40
17	Implementation	41
18	History	154
	Index	159