

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

May 6, 2021

Abstract

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

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

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

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

Remark: If you use LaTeX via Internet with, for example, Overleaf, you can upload the file `nicematrix.sty` in the repertory of your project in order to take full advantage of the latest version de `nicematrix`.¹

This package can be used with `xelatex`, `lualatex`, `pdflatex` but also by the classical workflow `latex-dvips-ps2pdf` (or Adobe Distiller). However, the file `nicematrix.dtx` of the present documentation should be compiled with XeLaTeX.

This package requires and **loads** the packages `l3keys2e`, `array`, `amsmath`, `pgfcore` and the module `shapes` of PGF (`tikz`, which is a layer over PGF is *not* loaded). The final user only has to load the package with `\usepackage{nicematrix}`.

The idea of `nicematrix` is to create PGF nodes under the cells and the positions of the rules of the tabular created by `array` and to use these nodes to develop new features. As usual with PGF, the coordinates of these nodes are written in the `.aux` to be used on the next compilation and that's why `nicematrix` may need **several compilations**.²

Most features of `nicematrix` may be used without explicit use of PGF or Tikz (which, in fact, is not loaded by default).

A command `\NiceMatrixOptions` is provided to fix the options (the scope of the options fixed by this command is the current TeX group: they are semi-global).

*This document corresponds to the version 5.15a of `nicematrix`, at the date of 2021/05/06.

¹The latest version of the file `nicematrix.sty` may be downloaded from the SVN server of TeXLive:
<https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty>

²If you use Overleaf, Overleaf will do automatically the right number of compilations.

1 The environments of this package

The package `nicematrix` defines the following new environments.

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

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

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

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

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

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

Important

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

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

2 The vertical space between the rows

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

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

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

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

The initial value of these parameters is 0 pt in order to have for the environments of `nicematrix` the same behaviour as those of `array` and `amsmath`. However, a value of 1 pt would probably be a good choice and we suggest to set them with `\NiceMatrixOptions`.³

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

³One should remark that these parameters apply also to the columns of type `S` of `siunitx` whereas the package `cellspace` is not able to act on such columns of type `S`.

3 The vertical position of the arrays

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

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

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

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

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

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

1. an item

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

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

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

1. an item

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

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

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

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

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

4 The blocks

4.1 General case

In the environments of `nicematrix`, it's possible to use the command `\Block` in order to place an element in the center of a rectangle of merged cells of the array.⁴

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

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

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

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

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

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

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

One may wish to raise the size of the “A” placed in the block of the previous example. Since this element is composed in math mode, it's not possible to use directly a command like `\large`, `\Large` and `\LARGE`. That's why the command `\Block` provides an option between angle brackets to specify some TeX code which will be inserted before the beginning of the math mode.⁵

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

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

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

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

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

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

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

⁴The spaces after a command `\Block` are deleted.

⁵This argument between angular brackets may also be used to insert a command of font such as `\bfseries` when the command `\\` is used in the content of the block.

- the key `fill` takes in as value a color and fills the block with that color;
- the key `draw` takes in as value a color and strokes the frame of the block with that color (the default value of that key is the current color of the rules of the array);
- the key `color` takes in as value a color and apply that color the content of the block but draws also the frame of the block with that color;
- the key `line-width` is the width (thickness) of the frame (this key should be used only when the key `draw` is in force);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt⁶);
- the key `borders` provides the ability to draw only some borders of the blocks; the value of that key is a (comma-separated) list of elements covered by `left`, `right`, `top` and `bottom`;
- **New 5.15** the keys `hvlines` draws all the vertical and horizontal rules in the block;
- **New 5.14** the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`).

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

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

```
\begin{NiceTabular}{cwc{2cm}wc{3cm}c}
rose      & tulipe & marguerite & dahlia \\
violette  &         &             &         \\
& \Block[draw=red,fill=[RGB]{204,204,255},rounded-corners]{2-2}
&         &         &         \\
& & & souci \\
pervenche & & & lys \\
arum      & iris & jacinthe & muguet
\end{NiceTabular}
```

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

4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

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

⁶This value is the initial value of the *rounded corners* of Tikz.

```

\begin{NiceTabular}{@{}>\bfseries\lr@{}} \hline
\Block{2-1}{John}      & 12 \\
                        & 13 \\ \hline
Steph                  & 8  \\ \hline
\Block{3-1}{Sarah}     & 18 \\
                        & 17 \\
                        & 15 \\ \hline
Ashley                 & 20 \\ \hline
Henry                  & 14 \\ \hline
\Block{2-1}{Madison}   & 15 \\
                        & 19 \\ \hline
\end{NiceTabular}

```

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

4.3 The mono-row blocks

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

4.4 The mono-cell blocks

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

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

- It's possible to use the command `\` in a (mono-cell) block.
- It's possible to use the option of horizontal alignment of the block in derogation of the type of column given in the preamble of the array.
- It's possible to draw a frame around the cell with the key `draw` of the command `\Block` and to fill the background with rounded corners with the keys `fill` and `rounded-corners`.⁷
- It's possible to draw one or several borders of the cell with the key `borders`.

```

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

```

Writer	year of birth
Hugo	1802
Balzac	1799

We recall that if the first mandatory argument of `\Block` is left blank, the block is mono-cell.⁸

4.5 A small remark

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

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

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

```

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

```

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

5 The rules

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

5.1 Some differences with the classical environments

5.1.1 The vertical rules

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

```

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

```

First	Second
Peter	
Mary	George

However, the vertical rules are not drawn in the blocks (created by `\Block`: cf. p. 4) nor in the corners (created by the key `corner`: cf. p. 9).

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

```

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

```

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

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

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

However, in this case, it is probably more clever to add a command `\OnlyMainNiceMatrix` (cf. p. 38):

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

5.1.2 The command `\cline`

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

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

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

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

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

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

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

5.2 The thickness and the color of the rules

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

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

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

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

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

If one wishes to define new specifiers for columns in order to draw vertical rules (for example with a specific color or thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 38.

5.3 The tools of `nicematrix` for the rules

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

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

All these tools don't draw the rules in the blocks nor in the empty corners (when the key `corners` is used).

- These blocks are:
 - the blocks created by the command `\Block`⁹ presented p. 4;
 - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `Vdots`, etc. (cf. p. 18).
- The corners are created by the key `corners` explained below (see p. 9).

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

5.3.1 The keys `hlines` and `vlines`

The keys `hlines` and `vlines` (which draw, of course, horizontal and vertical rules) take in as value a list of numbers which are the numbers of the rules to draw.

In fact, for the environments with delimiters (such as `{pNiceMatrix}` or `{bNiceArray}`), the key `vlines` don't draw the exterior rules (this is certainly the expected behaviour).

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

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

5.3.2 The key `hvlines`

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

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

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

5.3.3 The (empty) corners

The four `corners` of an array will be designed by `NW`, `SW`, `NE` and `SE` (*north west*, *south west*, *north east* and *south east*).

For each of these corners, we will call *empty corner* (or simply *corner*) the reunion of all the empty rectangles starting from the cell actually in the corner of the array.¹⁰

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

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

¹⁰For sake of completeness, we should also say that a cell contained in a block (even an empty cell) is not taken into account for the determination of the corners. That behaviour is natural.

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

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

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

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

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

5.4 The command `\diagbox`

```

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

```

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

¹¹The author of this document considers that type of construction as graphically poor.

5.5 Dotted rules

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

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

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

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

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

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

It's possible to change in `nicematrix` the letter used to specify a vertical dotted line with the option `letter-for-dotted-lines` available in `\NiceMatrixOptions`. Thus released, the letter “:” can be used otherwise (for example by the package `arydshln`¹²).

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 rule¹³. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

6 The color of the rows and columns

6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

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

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

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

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

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

The package `nicematrix` provides some tools (independent of `colortbl`) to draw the colored panels first, and, then, the content of the cells and the rules. This strategy is more conform to the “painting model” of the formats PostScript and PDF and is more suitable for the PDF viewers. However, it requires several compilations.¹⁴

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

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

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

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

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

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

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

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

```
\begin{NiceTabular}{|c|c|c|}
\CodeBefore
  \cellcolor[HTML]{FFFF88}{3-1,2-2,1-3}
\Body
\hline
a & b & c \\ \hline
e & f & g \\ \hline
h & i & j \\ \hline
\end{NiceTabular}
```

a	b	c
e	f	g
h	i	j

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

```
\begin{NiceTabular}{|c|c|c|}
\CodeBefore
  \rectanglecolor{blue!15}{2-2}{3-3}
\Body
\hline
a & b & c \\ \hline
e & f & g \\ \hline
h & i & j \\ \hline
\end{NiceTabular}
```

a	b	c
e	f	g
h	i	j

¹⁴If you use Overleaf, Overleaf will do automatically the right number of compilations.

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

- The command `\arraycolor` takes in as mandatory argument a color and color the whole tabular with that color (excepted the potential exterior rows and columns: cf. p. 17). It's only a particular case of `\rectanglecolor`.
- The command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

```

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

```

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

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

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

```

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

```

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

- The command `\columncolor` takes its name from the command `\columncolor` of `colortbl`. Its syntax is similar to the syntax of `\rowcolor`.
- The command `\rowcolors` (with a *s*) takes its name from the command `\rowcolors` of `xcolor`¹⁶. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows of the tabular with the tow colors (provided in second and third argument), beginning with the row whose number is given in first (mandatory) argument.

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

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

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

- The key `cols` describes a set of columns. The command `\rowcolors` will color only the cells of these columns. The value is a comma-separated list of intervals of the form i - j (where i or j may be replaced by $*$).
- With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.¹⁷
- With the key `respect-blocks` the “rows” alternately colored may extend over several rows if they have to incorporate blocks (created with the command `\Block`: cf. p. 4).

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

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

```
\begin{NiceTabular}{lrr}[hvlines]
\CodeBefore
  \rowcolors{1}{blue!10}{}[respect-blocks]
\Body
\Block{2-1}{John}      & 12 \\\
                        & 13 \\\
Steph                  & 8 \\\
\Block{3-1}{Sarah}     & 18 \\\
                        & 17 \\\
                        & 15 \\\
Ashley                 & 20 \\\
Henry                  & 14 \\\
\Block{2-1}{Madison}   & 15 \\\
                        & 19
\end{NiceTabular}
```

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

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

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

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

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

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

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

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

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

6.3 Color tools with the syntax of `colortbl`

It's possible to access the preceding tools with a syntax close to the syntax of `colortbl`. For that, one must use the key `colortbl-like` in the current environment.¹⁸

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

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

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

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

¹⁸As for now, this key is *not* available in `\NiceMatrixOptions`.

7 The width of the columns

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

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

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

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

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

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

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

It's possible to give the special value `auto` to the option `columns-width`: all the columns of the array will have a width equal to the widest cell of the array.¹⁹

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

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

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`²⁰. The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

¹⁹The 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).

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


```

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

```

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

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

8 The exterior rows and columns

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

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

```

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

```

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

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

We have several remarks to do.

- For the environments with an explicit preamble (i.e. `{NiceTabular}`, `{NiceArray}` and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type `r` for the first column and `l` for the last one.²¹
- One may wonder how `nicematrix` determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
 - For the environments with explicit preamble, like `{NiceTabular}` and `{pNiceArray}`, the number of columns can obviously be computed from the preamble.
 - When the option `light-syntax` (cf. p. 33) is used, `nicematrix` has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (but not the number of columns).

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

- In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

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

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

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

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

Remarks

- As shown in the previous example, the horizontal and vertical rules doesn't extend in the exterior rows and columns.
- However, if one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 38.
- A specification of color present in `code-for-first-row` also applies to a dotted line draw in this exterior "first row" (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 16) doesn't apply to the "first column" and "last column".
- For technical reasons, it's not possible to use the option of the command `\` after the "first row" or before the "last row". The placement of the delimiters would be wrong. If you are looking for a workaround, consider the command `\SubMatrix` in the `\CodeAfter` described p. 24.

9 The continuous dotted lines

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

`\vdots`, `\ddots` and `\iddots`.²²

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells²³ on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it's an horizontal line; for `\Vdots`, it's a vertical line and for `\Ddots` and `\Iddots` diagonal ones. It's possible to change the color of these lines with the option `color`.²⁴

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

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

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

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

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

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

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

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

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

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

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.²⁵

²²The 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`.

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

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

²⁵In `nicematrix`, one should use `\hspace*` and not `\hspace` for such an usage because `nicematrix` loads `array`. One may also remark that it's possible to fix the width of a column by using the environment `{NiceArray}` (or one of its variants) with a column of type `w` or `W`: see p. 16

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

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

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

9.1 The option `nullify-dots`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```


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


```

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`²⁶ is loaded (but you might have problem if you use `\rowcolor` on the same row as `\Hdotsfor`).

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

```

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

```

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

9.3 How to generate the continuous dotted lines transparently

Imagine you have a document with a great number of mathematical matrices with ellipsis. You may wish to use the dotted lines of `nicematrix` without having to modify the code of each matrix. It's possible with the keys. `renew-dots` and `renew-matrix`.²⁷

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

²⁷The options `renew-dots`, `renew-matrix` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage`. There is also a key `transparent` which is an alias for the conjunction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.

- The option `renew-dots`

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

- The option `renew-matrix`

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

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

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

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

9.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `\CodeAfter` which is described p. 24) accept two optional arguments specified by the tokens `_` and `^` for labels positionned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

```
$\begin{bNiceMatrix}
1 & & \hspace*{1cm} & & & & 0 & \ll[8mm]
& & \Ddots^{\text{times}} & & & & & \\
0 & & & & & & & 1 \\
\end{bNiceMatrix}$
```

$$\begin{bmatrix} 1 & & & & & & 0 \\ & \ddots & & & & & \\ & & \text{\scriptsize n lines} & & & & \\ & & & \ddots & & & \\ 0 & & & & & & 1 \end{bmatrix}$$

9.5 Customisation of the dotted lines

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

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

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

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

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

The option `xdots/color`

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

The option `xdots/shorten`

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

The option `xdots/line-style`

It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).²⁸

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

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

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

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

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

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

9.6 The dotted lines and the rules

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

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

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

²⁸The 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.

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

10 The `\CodeAfter`

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

For the legibility of the code, an alternative syntax is provided: it's possible to give the instructions of the `code-after` at the end of the environment, after the keyword `\CodeAfter`. Although `\CodeAfter` is a keyword, it takes in an optional argument (between square brackets). The keys accepted form a subset of the keys of the command `\WithArrowsOptions`.

The experienced users may, for instance, use the PGF/Tikz nodes created by `nicematrix` in the `\CodeAfter`. These nodes are described further beginning on p. 33.

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

10.1 The command `\line` in the `\CodeAfter`

The command `\line` draws directly dotted lines between nodes. It takes in two arguments for the two cells to link, both of the form $i-j$ where i is the number of the row and j is the number of the column. The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 22).

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

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

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

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

```
\begin{bNiceMatrix}
1      & \Cdots & 1      & 2      & \Cdots & 2      & \\
0      & \Ddots & \Vdots & \Vdots & \hspace*{2.5cm} & \Vdots & \\
\Vdots & \Ddots & & & & & \\
0      & \Cdots & 0 & 1      & 2      & \Cdots & 2 \\
\CodeAfter \line[shorten=6pt]{1-5}{4-7}
\end{bNiceMatrix}
```

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

10.2 The command `\SubMatrix` in the `\CodeAfter`

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

- the first argument is the left delimiter, which may be any extensible delimiter provided by LaTeX : `(`, `[`, `\{`, `\langle`, `\lgroup`, `\lfloor`, etc. but also the null delimiter `.`;

³⁰There is also a key `code-before` described p. 12.

- the second argument is the upper-left corner of the submatrix with the syntax $i-j$ where i the number of row and j the number of column;
- the third argument is the lower-right corner with the same syntax;
- the fourth argument is the right delimiter;
- the last argument, which is optional, is a list of key-value pairs.³¹

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

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

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

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

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

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

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

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

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

³¹There is no optional argument between square brackets in first position because a square bracket just after `\SubMatrix` must be interpreted as the first (mandatory) argument of the command `\SubMatrix`: that bracket is the left delimiter of the sub-matrix to construct (eg.: `\SubMatrix[{2-2}{4-7}]`).

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

```


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


```

There is also a key `name` which gives a name to the submatrix created by `\SubMatrix`. That name is used to create PGF/Tikz nodes: cf p. 37.

New 5.15 It's also possible to specify some delimiters³² by placing them in the preamble of the environment (for the environments with a preamble: `{NiceArray}`, `{pNiceArray}`, etc.). This syntax is inspired by the extension `blkarray`.

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

```


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


```

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

11 The notes in the tabulars

11.1 The footnotes

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

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

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

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

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

11.2 The notes of tabular

The package `nicematrix` also provides a command `\tabularnote` which gives the ability to specify notes that will be composed at the end of the array with a width of line equal to the width of the array (excepted the potential exterior columns). With no surprise, that command is available only in the environments without delimiters, that is to say `{NiceTabular}`, `{NiceArray}` and `{NiceMatrix}`. In fact, this command is available only if the extension `enumitem` has been loaded (before or after `nicematrix`). Indeed, the notes are composed at the end of the array with a type of list provided by the package `enumitem`.

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

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

For an illustration of some of those remarks, see table 1, p. 28. This table has been composed with the following code.

```

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

```

Table 1: Use of `\tabularnote`^a

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

Some text before the notes.

^a It's possible to put a note in the caption.

^b Considered as the first nurse of history.

^c Nicknamed “the Lady with the Lamp”.

^d The label of the note is overlapping.

11.3 Customisation of the tabular notes

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

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

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

```

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

```

We detail these keys.

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

Initial value: `false`

That key is also available within a given environment.

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

Initial value: `false`

That key is also available within a given environment.

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

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

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

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

Initial value: `#1`

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

```

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

```

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

Initial value: `#1`

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

```

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

```

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

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

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

This initial value contains the specification `align = left` which requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 28).

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

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

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

Initial value: `empty`

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

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

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

For an example of customisation of the tabular notes, see p. 40.

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

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

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

12 Other features

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

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

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

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

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

12.2 Alignment option in {NiceMatrix}

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

```


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


```

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}

```

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

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

```

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

```

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

12.4 The option small

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

```


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


```

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

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

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

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

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

12.5 The counters `iRow` and `jCol`

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column³³. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `code-before` (cf. p. 12) and in the `\CodeAfter` (cf. p. 24), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

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

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

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

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

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

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

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

³³We recall that the exterior “first row” (if it exists) has the number 0 and that the exterior “first column” (if it exists) has also the number 0.

12.6 The option `light-syntax`

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

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

```


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


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


```

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

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.³⁴

12.7 Color of the delimiters

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

```


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


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


```

12.8 The environment `{NiceArrayWithDelims}`

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

```


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


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


```

13 Use of Tikz with `nicematrix`

13.1 The nodes corresponding to the contents of the cells

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

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

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

³⁴The 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.

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

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

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

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

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

```

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

```

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

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

```

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

```

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

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

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

In fact, the package `nicematrix` can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option `create-medium-nodes` and the second ones with the option `create-large-nodes`.³⁶

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.

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

³⁶There 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.³⁷

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

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.³⁸

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

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

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

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

Here is an array composed with the following code:

```
\large
\begin{NiceTabular}{\w{2cm}\l{1}}[hvlines]
fraise & amande & abricot \\
prune & pêche & poire \\
noix & noisette & brugnon
\end{NiceTabular}
```

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

Here are the “large nodes” of this array (with-out use of `margin` nor `extra-margin`).

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

³⁸The options `left-margin` and `right-margin` take dimensions as values but, if no value is given, the default value is used, which is `\arraycolsep` (by default: 5 pt). There is also an option `margin` to fix both `left-margin` and `right-margin` to the same value.

13.3 The nodes which indicate the position of the rules

The package `nicematrix` creates a PGF/Tikz node merely called i (with the classical prefix) at the intersection of the horizontal rule of number i and the vertical rule of number i (more specifically the potential position of those rules because maybe there are not actually drawn). The last node has also an alias called `last`.

New 5.14 There is also a node called $i.5$ midway between the node i and the node $i + 1$. These nodes are available in the `\CodeBefore` and the `\CodeAfter`.

	tulipe	lys
arum		violette mauve
muguet	dahlia	

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

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

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

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

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

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

13.4 The nodes corresponding to the command `\SubMatrix`

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

If a command `\SubMatrix` has been used with the key `name` with an expression such as `name=MyName` three PGF/Tikz nodes are created with the names `MyName-left`, `MyName` and `MyName-right`.

The nodes `MyName-left` and `MyName-right` correspond to the delimiters left and right and the node `MyName` correspond to the submatrix itself.

In the following example, we have highlighted these nodes (the submatrix itself has been created with `\SubMatrix\{{2-2}{3-3}\}`).

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

14 API for the developpers

The package `nicematrix` provides two variables which are internal but public³⁹:

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

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

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

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

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

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

Here is an example of use:

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

Tokyo	Paris	London
Lima	Oslo	Miami
Los Angeles	Madrid	Roma

³⁹According to the LaTeX3 conventions, each variable with name beginning with `\g_nicematrix` ou `\l_nicematrix` is public and each variable with name beginning with `\g__nicematrix` or `\l__nicematrix` is private.

15 Technical remarks

15.1 Definition of new column types

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

For example, one may wish to define a new column type `?` in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job⁴⁰:

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

The heavy vertical rule won't extend in the exterior rows.⁴¹

```
\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 lines⁴² of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That's why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

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

Example with parallelization (default):

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

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

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

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

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

The same example without parallelization:

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

⁴⁰The command `\vrule` is a TeX (and not LaTeX) command.

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

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

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

15.3 The “empty” cells

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

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

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

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

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

15.4 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea⁴³. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`⁴⁴. The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

15.5 Incompatibilities

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

Anyway, in order to use `arydshln`, one must first free the letter “:” by giving a new letter for the vertical dotted rules of `nicematrix`:

```
\NiceMatrixOptions{letter-for-dotted-lines=;}
```

⁴³In 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).*

⁴⁴And 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.

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

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

16 Examples

16.1 Notes in the tabulars

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

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

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument⁴⁶

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

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

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

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

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

⁴⁶In fact: the value of its argument.


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

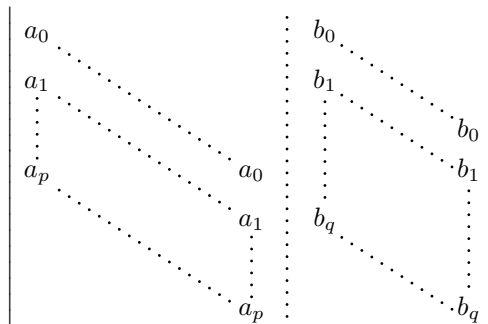
Last name	First name	Birth day
Achard*	Jacques	June 5, 2005
Lefebvre**	Mathilde	January 23, 1975
Vanesse	Stephany	October 30, 1994
Dupont	Chantal	January 15, 1998

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

16.2 Dotted lines

An example with the resultant of two polynoms:

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



An example for a linear system:

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

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

16.3 Dotted lines which are no longer dotted

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

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

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

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

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

16.4 Stacks of matrices

We often need to compose mathematical matrices on top on each other (for example for the resolution of linear systems).

In order to have the columns aligned one above the other, it's possible to fix a width for all the columns. That's what is done in the following example with the environment `{NiceMatrixBlock}` and its option `auto-columns-width`.

```

\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions
{
  light-syntax,
  last-col, code-for-last-col = \color{blue} \scriptstyle,
}
\setlength{\extrarowheight}{1mm}

$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 {} ;
3 -18 12 1 4 ;
-3 -46 29 -2 -15 ;
9 10 -5 4 7
\end{pNiceArray}$

\smallskip
$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 ;
0 64 -41 1 19 { L_2 \gets L_1-4L_2 } ;
0 -192 123 -3 -57 { L_3 \gets L_1+4L_3 } ;
0 -64 41 -1 -19 { L_4 \gets 3L_1-4L_4 } ;
\end{pNiceArray}$

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

\smallskip
$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 {} ;
0 64 -41 1 19 ;
\end{pNiceArray}$

\end{NiceMatrixBlock}

```

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

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

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

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

However, one can see that the last matrix is not perfectly aligned with others. That's why, in LaTeX, the parenthesis have not exactly the same width (smaller parenthesis are a bit slimer).

In order to solve that problem, it's possible to require the delimiters to be composed with the maximal width, thanks to the boolean key `delimiters/max-width`.

```

\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions
{
  delimiters/max-width,
  light-syntax,
  last-col, code-for-last-col = \color{blue}\scriptstyle,
}
\setlength{\extrarowheight}{1mm}

$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 {} ;
3 -18 12 1 4 ;
-3 -46 29 -2 -15 ;
9 10 -5 4 7
\end{pNiceArray}$

...
\end{NiceMatrixBlock}

```

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

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

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

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

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

```

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

```

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

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

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

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

16.5 How to highlight cells of a matrix

In order to highlight a cell of a matrix, it's possible to “draw” that cell with the key **draw** of the command `\Block` (this is one of the uses of a mono-cell block⁴⁷).

```

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

```

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

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

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don't spread the cells of the array. We recall that, on the other side, the command `\hline`, the specifier “|” and the options `hlines`, `vlines` and `hvlines` spread the cells.⁴⁸

It's possible to color a row with `\rowcolor` in the *code-before* (or with `\rowcolor` in the first cell of the row if the key `colortbl-like` is used—even when `colortbl` is not loaded).

```
\begin{pNiceArray}{>{\strut}cccc}[margin, extra-margin=2pt,colortbl-like]
  \rowcolor{red!15}A_{11} & A_{12} & A_{13} & A_{14} \\
  A_{21} & \rowcolor{red!15}A_{22} & A_{23} & A_{24} \\
  A_{31} & A_{32} & \rowcolor{red!15}A_{33} & A_{34} \\
  A_{41} & A_{42} & A_{43} & \rowcolor{red!15}A_{44}
\end{pNiceArray}
```

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

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

Caution : Some PDF readers are not able to show transparency.⁴⁹

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

```
\usepackage{tikz}
\usetikzlibrary{fit}
```

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

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

⁴⁸For the command `\cline`, see the remark p. 8.

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

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

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

```

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

```

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

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

```

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

```

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

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

```

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

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

```

We obtain the following matrix.

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

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

```

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

```

We obtain the following matrix.

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

16.6 Utilisation of `\SubMatrix` in the `\CodeBefore`

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

The whole figure is an environment `{NiceArray}` and the three pairs of parenthesis have been added with `\SubMatrix` in the `code-before`.

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

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

17 Implementation

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

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

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

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

Declaration of the package and packages loaded

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

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

<@@=nicematrix>

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

```

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

```


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

```

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

```

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

We load some packages. The package `xparse` is still loaded for use on Overleaf.

```

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

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

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

```

Technical definitions

```

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

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

```

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

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

```

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

```

```

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

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date January 2021, the current version `revtex4-2` is 4.2e (compatible with `booktabs`).

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

87 } }

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

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

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

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

Idem for `\CT@drs@`.

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

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

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

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

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

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

```
131   \everycr { }
```

⁵⁰See question 99041 on TeX StackExchange.

```

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

```

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

```

135 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

The following command is a small shortcut.

```

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

```

```

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

```

```

163 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

```

The column S of siunitx

We want to know whether the package `siunitx` is loaded and, if it is loaded, we redefine the S columns of `siunitx`.

```

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

```

170 }

The command `\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:`.

```
171 \cs_set_protected:Npn \@@_adapt_S_column:
172 {
173   \bool_if:NT \c_@@_siunitx_loaded_bool
174   {
175     \group_begin:
176     \@temptokena = { }
```

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

```
177   \cs_set_eq:NN \NC@find \prg_do_nothing:
178   \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`).

```
179   \tl_gset:NV \g_tmpa_tl \@temptokena
180   \group_end:
181   \tl_new:N \c_@@_table_collect_begin_tl
182   \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
183   \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
184   \tl_new:N \c_@@_table_print_tl
185   \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).

```
186   \cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
```

```

187     }
188 }

```

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

```

189 \AtBeginDocument
190 {
191     \bool_if:nTF { ! \c_@@_siunitx_loaded_bool }
192     { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
193     {
194         \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
195         {
196             \renewcommand*{\NC@rewrite@S}[1] []
197             {
198                 \@temptokena \exp_after:wN
199                 {
200                     \tex_the:D \@temptokena
201                     > { \@@_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.

```

202             \@@_true_c:
203             < { \c_@@_table_print_tl \@@_end_Cell: }
204         }
205         \NC@find
206     }
207 }
208 }
209 }

```

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

```

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

```

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

```

211 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
212 {
213     \iow_now:Nn \@mainaux
214     {
215         \ExplSyntaxOn
216         \cs_if_free:NT \pgfsyspdfmark
217         { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
218         \ExplSyntaxOff
219     }
220     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
221 }

```

Parameters

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

```

222 \bool_new:N \c_@@_define_L_C_R_bool

```

```

223 \cs_new_protected:Npn \@@_define_L_C_R:
224 {
225     \newcolumntype L l
226     \newcolumntype C c
227     \newcolumntype R r
228 }

```

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.

```

229 \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).

```

230 \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.

```

231 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
232 { \int_use:N \g_@@_env_int }

```

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

```

233 \cs_new_protected:Npn \@@_qpoint:n #1
234 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

235 \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).

```

236 \dim_new:N \l_@@_columns_width_dim

```

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

```

237 \int_new:N \g_@@_row_total_int
238 \int_new:N \g_@@_col_total_int

```

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

```

239 \tl_new:N \l_@@_cell_type_tl
240 \tl_set:Nn \l_@@_cell_type_tl { c }

```

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

```

241 \dim_new:N \g_@@_blocks_wd_dim

```

Idem pour the mono-row blocks.

```

242 \dim_new:N \g_@@_blocks_ht_dim
243 \dim_new:N \g_@@_blocks_dp_dim

```

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

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

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

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

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

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

In fact, if there is delimiters in the preamble of `{NiceArray}` (eg: `[cccc]`), this boolean will be set to false.

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

```
247 \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.

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

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

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

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

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

```
251 \cs_new_protected:Npn \@@_test_if_math_mode:
252 {
253   \if_mode_math: \else:
254     \@@_fatal:n { Outside-math-mode }
255   \fi:
256 }
```

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

```
257 \tl_new:N \l_@@_letter_vlism_tl
```

The list of the columns where vertical lines in sub-matrices (`vlism`) must be drawn. Of course, the actual value of this sequence will be known after the analyse of the preamble of the array.

```
258 \seq_new:N \g_@@_cols_vlism_seq
```

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

```
259 \colorlet { nicematrix-last-col } { . }
260 \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*).

```
261 \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*.

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


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

```

263 \cs_new:Npn \@@_full_name_env:
264 {
265   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
266   { command \space \c_backslash_str \g_@@_name_env_str }
267   { environment \space \{ \g_@@_name_env_str \} }
268 }
```

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

```

269 \tl_new:N \g_nicematrix_code_after_tl
```

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

```

270 \tl_new:N \l_@@_code_tl
```

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

```

271 \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.

```

272 \int_new:N \l_@@_old_iRow_int
273 \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.

```

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

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

```

275 \bool_new:N \g_@@_after_col_zero_bool
```

A kind of false row will be inserted at the end of the array for the construction of the `col` nodes (and also to fix the width of the columns when `columns-width` is used). When this special row will be created, we will raise the flag `\g_@@_row_of_col_done_bool` in order to avoid some actions set in the redefinition of `\everycr` when the last `\cr` of the `\halign` will occur (after that row of `col` nodes).

```

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

It's possible to use the command `\NotEmpty` to specify explicitly that a cell must be considered as non empty by `nicematrix` (the Tikz nodes are constructed only in the non empty cells).

```

277 \bool_new:N \g_@@_not_empty_cell_bool
```

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

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

- The final user can explicitly add material in `\l_@@_code_before_tl` by using the key `code-before` or the keyword `\CodeBefore` (with the keyword `\Body`).

```
278 \tl_new:N \l_@@_code_before_tl
279 \bool_new:N \l_@@_code_before_bool
```

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

```
280 \dim_new:N \l_@@_x_initial_dim
281 \dim_new:N \l_@@_y_initial_dim
282 \dim_new:N \l_@@_x_final_dim
283 \dim_new:N \l_@@_y_final_dim
```

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

```
284 \dim_zero_new:N \l_tmpc_dim
285 \dim_zero_new:N \l_tmpd_dim
```

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

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

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

```
287 \dim_new:N \g_@@_width_last_col_dim
288 \dim_new:N \g_@@_width_first_col_dim
```

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces:

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

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

```
289 \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}`.

```
290 \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}`.

```
291 \seq_new:N \g_@@_pos_of_xdots_seq
```

The sequence `\g_@@_pos_of_xdots_seq` will be used when we will draw the rules required by the key `hvlines` (these rules won't be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key `draw=red!15` when using the command `\Block`). In that case, the rules specified, for instance, by `hvlines` must not be drawn around the block. That's why we keep the information of all that stroken blocks in the following sequence.

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

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

```
293 \seq_new:N \l_@@_corners_cells_seq
```

The list of the names of the potential `\SubMatrix` in the `\CodeAfter` of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given `\SubMatrix`).

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

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

```
295 \int_new:N \l_@@_row_min_int
296 \int_new:N \l_@@_row_max_int
297 \int_new:N \l_@@_col_min_int
298 \int_new:N \l_@@_col_max_int
```

The following sequence will be used when the command `\SubMatrix` is used in the `code-before` (and not in the `\CodeAfter`). It will contain the position of all the sub-matrices specified in the `code-before`. Each sub-matrix is represented by an “object” of the forme $\{i\}\{j\}\{k\}\{l\}$ where i and j are the number of row and column of the upper-left cell and k and l the number of row and column of the lower-right cell.

```
299 \seq_new:N \g_@@_submatrix_seq
```

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

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

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

```
301 \tl_new:N \l_@@_fill_tl
302 \tl_new:N \l_@@_draw_tl
303 \clist_new:N \l_@@_borders_clist
304 \dim_new:N \l_@@_rounded_corners_dim
```

The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by `nicematrix` when the key `corners` is used).

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

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

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

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

The parameters of position of the label of a block. For the horizontal position, the possible values are `c`, `r` and `l`. For the vertical position, the possible values are `c`, `t` and `b`. Of course, it would be interesting to program a key `T` and a key `B`.

```
307 \tl_new:N \l_@@_hpos_of_block_tl
308 \tl_set:Nn \l_@@_hpos_of_block_tl { c }
309 \tl_new:N \l_@@_vpos_of_block_tl
310 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

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

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

```
312 \bool_new:N \l_@@_hvlines_block_bool
```

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

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

```

314 \dim_new:N \l_@@_submatrix_extra_height_dim
315 \dim_new:N \l_@@_submatrix_left_xshift_dim
316 \dim_new:N \l_@@_submatrix_right_xshift_dim
317 \clist_new:N \l_@@_hlines_clist
318 \clist_new:N \l_@@_vlines_clist
319 \clist_new:N \l_@@_submatrix_hlines_clist
320 \clist_new:N \l_@@_submatrix_vlines_clist

```

Variables for the exterior rows and columns

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

• First row

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

```

321 \int_new:N \l_@@_first_row_int
322 \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.

```

323 \int_new:N \l_@@_first_col_int
324 \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).

```

325 \int_new:N \l_@@_last_row_int
326 \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”.⁵¹

```

327 \bool_new:N \l_@@_last_row_without_value_bool

```

Idem for `\l_@@_last_col_without_value_bool`

```

328 \bool_new:N \l_@@_last_col_without_value_bool

```

• Last column

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

```

329 \int_new:N \l_@@_last_col_int
330 \int_set:Nn \l_@@_last_col_int { -2 }

```

⁵¹We 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.

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.

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

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

The command `\tablarnote`

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

```
332 \newcounter { tablarnote }
```

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

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

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

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

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).

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

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

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

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

```
337 \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`.

```
338 \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`.

```
339 \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`).

```

340 \AtBeginDocument
341 {
342   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
343   {
344     \NewDocumentCommand \tabularnote { m }
345     { \@@_error:n { enumitem~not~loaded } }
346   }
347   {

```

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.

```

348   \newlist { tabularnotes } { enumerate } { 1 }
349   \setlist [ tabularnotes ]
350   {
351     topsep = 0pt ,
352     noitemsep ,
353     leftmargin = * ,
354     align = left ,
355     labelsep = 0pt ,
356     label =
357     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
358   }
359   \newlist { tabularnotes* } { enumerate* } { 1 }
360   \setlist [ tabularnotes* ]
361   {
362     afterlabel = \nobreak ,
363     itemjoin = \quad ,
364     label =
365     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
366   }

```

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.⁵²

```

367   \NewDocumentCommand \tabularnote { m }
368   {
369     \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
370     { \@@_error:n { tabularnote~forbidden } }
371     {

```

`\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 theses notes as a comma separated list (e.g. a,b,c).

```

372     \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`.

```

373     \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
374     \peek_meaning:NF \tabularnote
375     {

```

⁵²We should try to find a solution to that problem.

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.

```
376         \hbox_set:Nn \l_tmpa_box
377         {
```

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

```
378         \@@_notes_label_in_tabular:n
379         {
380             \stepcounter { tabularnote }
381             \@@_notes_style:n { tabularnote }
382             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
383             {
384                 ,
385                 \stepcounter { tabularnote }
386                 \@@_notes_style:n { tabularnote }
387             }
388         }
389     }
```

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.

```
390         \addtocounter { tabularnote } { -1 }
391         \refstepcounter { tabularnote }
392         \int_zero:N \l_@@_number_of_notes_int
393         \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).

```
394         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
395     }
396 }
397 }
398 }
399 }
```

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.

```
400 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
401 {
402     \begin { pgfscope }
403     \pgfset
404     {
405         outer~sep = \c_zero_dim ,
406         inner~sep = \c_zero_dim ,
407         minimum~size = \c_zero_dim
408     }
409     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
410     \pgfnode
411     { rectangle }
412     { center }
413     {
414         \vbox_to_ht:nn
415         { \dim_abs:n { #5 - #3 } }
```

```

416         {
417             \vfill
418             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
419         }
420     }
421     { #1 }
422     { }
423 \end { pgfscope }
424 }

```

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

```

425 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
426 {
427     \begin { pgfscope }
428     \pgfset
429     {
430         outer-sep = \c_zero_dim ,
431         inner-sep = \c_zero_dim ,
432         minimum-size = \c_zero_dim
433     }
434     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
435     \pgfpointdiff { #3 } { #2 }
436     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
437     \pgfnode
438     { rectangle }
439     { center }
440     {
441         \vbox_to_ht:nn
442         { \dim_abs:n \l_tmpb_dim }
443         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
444     }
445     { #1 }
446     { }
447 \end { pgfscope }
448 }

```

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.

```

449 \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`.

```

450 \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`).

```

451 \dim_new:N \l_@@_cell_space_top_limit_dim
452 \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.

```

453 \dim_new:N \l_@@_inter_dots_dim
454 \AtBeginDocument { \dim_set:Nn \l_@@_inter_dots_dim { 0.45 em } }

```


The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even though it is obsolete).

The following dimension is the minimal distance between a node (in fact an anchor of that node) and a dotted line (we say “minimal” because, by definition, a dotted line is not a continuous line and, therefore, this distance may vary a little).

```
455 \dim_new:N \l_@@_xdots_shorten_dim
456 \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 em } }
```

The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even though it is obsolete).

The following dimension is the radius of the dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.53 pt but it will be changed if the option `small` is used.

```
457 \dim_new:N \l_@@_radius_dim
458 \AtBeginDocument { \dim_set:Nn \l_@@_radius_dim { 0.53 pt } }
```

The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even if it is obsolete).

The token list `\l_@@_xdots_line_style_tl` corresponds to the option `tikz` of the commands `\Cdots`, `\Ldots`, etc. and of the options `line-style` for the environments and `\NiceMatrixOptions`. The constant `\c_@@_standard_tl` will be used in some tests.

```
459 \tl_new:N \l_@@_xdots_line_style_tl
460 \tl_const:Nn \c_@@_standard_tl { standard }
461 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

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

The string `\l_@@_baseline_tl` may contain one of the three values `t`, `c` or `b` as in the option of the environment `{array}`. However, it may also contain an integer (which represents the number of the row to which align the array).

```
463 \tl_new:N \l_@@_baseline_tl
464 \tl_set:Nn \l_@@_baseline_tl c
```

The flag `\l_@@_exterior_arraycolsep_bool` corresponds to the option `exterior-arraycolsep`. If this option is set, a space equal to `\arraycolsep` will be put on both sides of an environment `{NiceArray}` (as it is done in `{array}` of `array`).

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

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

```
466 \bool_new:N \l_@@_parallelize_diags_bool
467 \bool_set_true:N \l_@@_parallelize_diags_bool
```

The following parameter correspond to the key `corners`. The elements of that `clist` must be in NW, SW, NE and SE.

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

```
469 \dim_new:N \l_@@_notes_above_space_dim
470 \AtBeginDocument { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even if it is obsolete).

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

```
471 \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).

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

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

```
473 \bool_new:N \g_@@_recreate_cell_nodes_bool
```

The string `\l_@@_name_str` will contain the optional name of the environment: this name can be used to access to the Tikz nodes created in the array from outside the environment.

```
474 \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”.

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

```
476 \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`).

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

```
478 \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`.

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

```
480 \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.

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

```
482 \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 “:”.

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

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

```
484 \tl_new:N \l_@@_delimiters_color_tl
```

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option `columns-width` or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is fonction of its size. That’s why we create an option called `delimiters/max-width` which will give to the delimiters the width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

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

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

```
487 {
```

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

```
489   {
```

```
490     \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.

```

491     { \cs_if_exist_p:N \tikzpicture }
492     { \str_if_eq_p:nn { #1 } { standard } }
493     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
494     { \@@_error:n { bad-option-for-line-style } }
495   } ,
496   line-style .value_required:n = true ,
497   color .tl_set:N = \l_@@_xdots_color_tl ,
498   color .value_required:n = true ,
499   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
500   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 `_`.

```

501   down .tl_set:N = \l_@@_xdots_down_tl ,
502   up .tl_set:N = \l_@@_xdots_up_tl ,

```

The key `draw-first`, which is meant to be used only with `\Ddots` and `\Iddots`, which be caught when `\Ddots` or `\Iddots` is used (during the construction of the array and not when we draw the dotted lines).

```

503   draw-first .code:n = \prg_do_nothing: ,
504   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
505 }

```

```

506 \keys_define:nn { NiceMatrix / rules }
507 {
508   color .tl_set:N = \l_@@_rules_color_tl ,
509   color .value_required:n = true ,
510   width .dim_set:N = \arrayrulewidth ,
511   width .value_required:n = true
512 }

```

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

```

513 \keys_define:nn { NiceMatrix / Global }
514 {
515   delimiters .code:n =
516     \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
517   delimiters .value_required:n = true ,
518   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
519   rules .value_required:n = true ,
520   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
521   standard-cline .default:n = true ,
522   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
523   cell-space-top-limit .value_required:n = true ,
524   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
525   cell-space-bottom-limit .value_required:n = true ,
526   cell-space-limits .meta:n =
527     {
528       cell-space-top-limit = #1 ,
529       cell-space-bottom-limit = #1 ,
530     } ,
531   cell-space-limits .value_required:n = true ,
532   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
533   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
534   light-syntax .default:n = true ,
535   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
536   end-of-row .value_required:n = true ,
537   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
538   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
539   last-row .int_set:N = \l_@@_last_row_int ,

```

```

540 last-row .default:n = -1 ,
541 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
542 code-for-first-col .value_required:n = true ,
543 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
544 code-for-last-col .value_required:n = true ,
545 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
546 code-for-first-row .value_required:n = true ,
547 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
548 code-for-last-row .value_required:n = true ,
549 hlines .clist_set:N = \l_@@_hlines_clist ,
550 vlines .clist_set:N = \l_@@_vlines_clist ,
551 hlines .default:n = all ,
552 vlines .default:n = all ,
553 vlines-in-sub-matrix .code:n =
554 {
555   \tl_if_single_token:nTF { #1 }
556   { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
557   { @@_error:n { One-letter-allowed } }
558 } ,
559 vlines-in-sub-matrix .value_required:n = true ,
560 hvlines .code:n =
561 {
562   \clist_set:Nn \l_@@_vlines_clist { all }
563   \clist_set:Nn \l_@@_hlines_clist { all }
564 } ,
565 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.

```

566 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
567 renew-dots .value_forbidden:n = true ,
568 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
569 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
570 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
571 create-extra-nodes .meta:n =
572 { create-medium-nodes , create-large-nodes } ,
573 left-margin .dim_set:N = \l_@@_left_margin_dim ,
574 left-margin .default:n = \arraycolsep ,
575 right-margin .dim_set:N = \l_@@_right_margin_dim ,
576 right-margin .default:n = \arraycolsep ,
577 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
578 margin .default:n = \arraycolsep ,
579 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
580 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
581 extra-margin .meta:n =
582 { extra-left-margin = #1 , extra-right-margin = #1 } ,
583 extra-margin .value_required:n = true ,
584 }

```

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

```

585 \keys_define:nn { NiceMatrix / Env }
586 {
587

```

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

```

588 hvlines-except-corners .code:n =
589 {
590   \clist_set:Nn \l_@@_corners_clist { #1 }
591   \clist_set:Nn \l_@@_vlines_clist { all }
592   \clist_set:Nn \l_@@_hlines_clist { all }
593 } ,

```

```

594 hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
595 corners .clist_set:N = \l_@@_corners_clist ,
596 corners .default:n = { NW , SW , NE , SE } ,
597 code-before .code:n =
598 {
599   \tl_if_empty:nF { #1 }
600   {
601     \tl_put_right:Nn \l_@@_code_before_tl { #1 }
602     \bool_set_true:N \l_@@_code_before_bool
603   }
604 } ,

```

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

```

605 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
606 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
607 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
608 baseline .tl_set:N = \l_@@_baseline_tl ,
609 baseline .value_required:n = true ,
610 columns-width .code:n =
611   \tl_if_eq:nnTF { #1 } { auto }
612   { \bool_set_true:N \l_@@_auto_columns_width_bool }
613   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
614 columns-width .value_required:n = true ,
615 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.

```

616 \legacy_if:nF { measuring@ }
617 {
618   \str_set:Nn \l_tmpa_str { #1 }
619   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
620   { \@@_error:nn { Duplicate~name } { #1 } }
621   { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
622   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
623 } ,
624 name .value_required:n = true ,
625 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
626 code-after .value_required:n = true ,
627 colortbl-like .code:n =
628   \bool_set_true:N \l_@@_colortbl_like_bool
629   \bool_set_true:N \l_@@_code_before_bool ,
630 colortbl-like .value_forbidden:n = true
631 }
632 \keys_define:nn { NiceMatrix / notes }
633 {
634   para .bool_set:N = \l_@@_notes_para_bool ,
635   para .default:n = true ,
636   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
637   code-before .value_required:n = true ,
638   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
639   code-after .value_required:n = true ,
640   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
641   bottomrule .default:n = true ,
642   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
643   style .value_required:n = true ,
644   label-in-tabular .code:n =
645     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
646   label-in-tabular .value_required:n = true ,
647   label-in-list .code:n =
648     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
649   label-in-list .value_required:n = true ,
650   enumitem-keys .code:n =

```

```

651 {
652   \bool_if:NTF \c_@@_in_preamble_bool
653   {
654     \AtBeginDocument
655     {
656       \bool_if:NT \c_@@_enumitem_loaded_bool
657       { \setlist* [ tabularnotes ] { #1 } }
658     }
659   }
660   {
661     \bool_if:NT \c_@@_enumitem_loaded_bool
662     { \setlist* [ tabularnotes ] { #1 } }
663   }
664 },
665 enumitem-keys .value_required:n = true ,
666 enumitem-keys-para .code:n =
667 {
668   \bool_if:NTF \c_@@_in_preamble_bool
669   {
670     \AtBeginDocument
671     {
672       \bool_if:NT \c_@@_enumitem_loaded_bool
673       { \setlist* [ tabularnotes* ] { #1 } }
674     }
675   }
676   {
677     \bool_if:NT \c_@@_enumitem_loaded_bool
678     { \setlist* [ tabularnotes* ] { #1 } }
679   }
680 },
681 enumitem-keys-para .value_required:n = true ,
682 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
683 }
684 \keys_define:nn { NiceMatrix / delimiters }
685 {
686   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
687   max-width .default:n = true ,
688   color .tl_set:N = \l_@@_delimiters_color_tl ,
689   color .value_required:n = true ,
690 }

```

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

```

691 \keys_define:nn { NiceMatrix }
692 {
693   NiceMatrixOptions .inherit:n =
694   { NiceMatrix / Global } ,
695   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
696   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
697   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
698   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
699   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
700   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
701   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
702   NiceMatrix .inherit:n =
703   {
704     NiceMatrix / Global ,
705     NiceMatrix / Env ,
706   } ,
707   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
708   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
709   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,

```

```

710 NiceTabular .inherit:n =
711 {
712     NiceMatrix / Global ,
713     NiceMatrix / Env
714 } ,
715 NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
716 NiceTabular / rules .inherit:n = NiceMatrix / rules ,
717 NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
718 NiceArray .inherit:n =
719 {
720     NiceMatrix / Global ,
721     NiceMatrix / Env ,
722 } ,
723 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
724 NiceArray / rules .inherit:n = NiceMatrix / rules ,
725 NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
726 pNiceArray .inherit:n =
727 {
728     NiceMatrix / Global ,
729     NiceMatrix / Env ,
730 } ,
731 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
732 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
733 pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
734 }

```

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

```

735 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
736 {
737     last-col .code:n = \tl_if_empty:nF { #1 }
738         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
739         \int_zero:N \l_@@_last_col_int ,
740     small .bool_set:N = \l_@@_small_bool ,
741     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.

```

742 renew-matrix .code:n = \@@_renew_matrix: ,
743 renew-matrix .value_forbidden:n = true ,

```

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

```

744 transparent .code:n =
745 {
746     \@@_renew_matrix:
747     \bool_set_true:N \l_@@_renew_dots_bool
748     \@@_error:n { Key~transparent }
749 } ,
750 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}`.

```

751 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.

```

752 columns-width .code:n =
753 \tl_if_eq:nnTF { #1 } { auto }
754 { \@@_error:n { Option~auto~for~columns~width } }
755 { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

756   allow-duplicate-names .code:n =
757     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
758   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`).

```

759   letter-for-dotted-lines .code:n =
760     {
761       \tl_if_single_token:nTF { #1 }
762         { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
763         { \@@_error:n { One-letter~allowed } }
764     } ,
765   letter-for-dotted-lines .value_required:n = true ,
766   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
767   notes .value_required:n = true ,
768   sub-matrix .code:n =
769     \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
770   sub-matrix .value_required:n = true ,
771   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
772 }
773 \str_new:N \l_@@_letter_for_dotted_lines_str
774 \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.

```

775 \NewDocumentCommand \NiceMatrixOptions { m }
776 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

777 \keys_define:nn { NiceMatrix / NiceMatrix }
778 {
779   last-col .code:n = \tl_if_empty:nTF {#1}
780     {
781       \bool_set_true:N \l_@@_last_col_without_value_bool
782       \int_set:Nn \l_@@_last_col_int { -1 }
783     }
784     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
785   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
786   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
787   small .bool_set:N = \l_@@_small_bool ,
788   small .value_forbidden:n = true ,
789   unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
790 }

```

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

```

791 \keys_define:nn { NiceMatrix / NiceArray }
792 {

```

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.

```

793   small .bool_set:N = \l_@@_small_bool ,
794   small .value_forbidden:n = true ,

```



```

795 last-col .code:n = \tl_if_empty:nF { #1 }
796         { \@@_error:n { last-col~non-empty~for~NiceArray } }
797         \int_zero:N \l_@@_last_col_int ,
798 notes / para .bool_set:N = \l_@@_notes_para_bool ,
799 notes / para .default:n = true ,
800 notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
801 notes / bottomrule .default:n = true ,
802 tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
803 tabularnote .value_required:n = true ,
804 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
805 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
806 unknown .code:n = \@@_error:n { Unknown~option~for~NiceArray }
807 }

808 \keys_define:nn { NiceMatrix / pNiceArray }
809 {
810 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
811 last-col .code:n = \tl_if_empty:nF {#1}
812         { \@@_error:n { last-col~non-empty~for~NiceArray } }
813         \int_zero:N \l_@@_last_col_int ,
814 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
815 small .bool_set:N = \l_@@_small_bool ,
816 small .value_forbidden:n = true ,
817 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
818 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
819 unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
820 }

```

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

```

821 \keys_define:nn { NiceMatrix / NiceTabular }
822 {
823 notes / para .bool_set:N = \l_@@_notes_para_bool ,
824 notes / para .default:n = true ,
825 notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
826 notes / bottomrule .default:n = true ,
827 tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
828 tabularnote .value_required:n = true ,
829 last-col .code:n = \tl_if_empty:nF {#1}
830         { \@@_error:n { last-col~non-empty~for~NiceArray } }
831         \int_zero:N \l_@@_last_col_int ,
832 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
833 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
834 unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular }
835 }

```

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}).

```

836 \cs_new_protected:Npn \@@_Cell:
837 {

```

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

```

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

```

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

```

839 \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.

```
840 \int_compare:nNnT \c@jCol = 1
841 { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
```

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

```
842 \hbox_set:Nw \l_@@_cell_box
843 \bool_if:NF \l_@@_NiceTabular_bool
844 {
845   \c_math_toggle_token
846   \bool_if:NT \l_@@_small_bool \scriptstyle
847 }
```

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.

```
848 \int_compare:nNnTF \c@iRow = 0
849 {
850   \int_compare:nNnT \c@jCol > 0
851   {
852     \l_@@_code_for_first_row_tl
853     \xglobal \colorlet { nicematrix-first-row } { . }
854   }
855 }
856 {
857   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
858   {
859     \l_@@_code_for_last_row_tl
860     \xglobal \colorlet { nicematrix-last-row } { . }
861   }
862 }
863 }
```

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.

```
864 \cs_new_protected:Npn \@@_begin_of_row:
865 {
866   \int_gincr:N \c@iRow
867   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
868   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
869   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
870   \pgfpicture
871   \pgfrememberpicturepositiononpagetrue
872   \pgfcoordinate
873   { \@@_env: - row - \int_use:N \c@iRow - base }
874   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
875   \str_if_empty:NF \l_@@_name_str
876   {
877     \pgfnodealias
878     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
879     { \@@_env: - row - \int_use:N \c@iRow - base }
880   }
881   \endpgfpicture
882 }
```

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.

```

883 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
884 {
885   \int_compare:nNnTF \c@iRow = 0
886   {
887     \dim_gset:Nn \g_@@_dp_row_zero_dim
888     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
889     \dim_gset:Nn \g_@@_ht_row_zero_dim
890     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
891   }
892   {
893     \int_compare:nNnT \c@iRow = 1
894     {
895       \dim_gset:Nn \g_@@_ht_row_one_dim
896       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
897     }
898   }
899 }

900 \cs_new_protected:Npn \@@_rotate_cell_box:
901 {
902   \box_rotate:Nn \l_@@_cell_box { 90 }
903   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
904   {
905     \vbox_set_top:Nn \l_@@_cell_box
906     {
907       \vbox_to_zero:n { }
908       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
909       \box_use:N \l_@@_cell_box
910     }
911   }
912   \bool_gset_false:N \g_@@_rotate_bool
913 }

914 \cs_new_protected:Npn \@@_adjust_size_box:
915 {
916   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
917   {
918     \box_set_wd:Nn \l_@@_cell_box
919     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
920     \dim_gzero:N \g_@@_blocks_wd_dim
921   }
922   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
923   {
924     \box_set_dp:Nn \l_@@_cell_box
925     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
926     \dim_gzero:N \g_@@_blocks_dp_dim
927   }
928   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
929   {
930     \box_set_ht:Nn \l_@@_cell_box
931     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
932     \dim_gzero:N \g_@@_blocks_ht_dim
933   }
934 }

935 \cs_new_protected:Npn \@@_end_Cell:
936 {
937   \@@_math_toggle_token:
938   \hbox_set_end:
939   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
940   \@@_adjust_size_box:
941   \box_set_ht:Nn \l_@@_cell_box
942   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
943   \box_set_dp:Nn \l_@@_cell_box
944   { \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”).

```

945 \dim_gset:Nn \g_@@_max_cell_width_dim
946 { \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”.

```

947 \@@_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 for now, we use the following technic:

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

```

948 \bool_if:NTF \g_@@_empty_cell_bool
949 { \box_use_drop:N \l_@@_cell_box }
950 {
951   \bool_lazy_or:nnTF
952     \g_@@_not_empty_cell_bool
953     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
954     \@@_node_for_cell:
955     { \box_use_drop:N \l_@@_cell_box }
956 }
957 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
958 \bool_gset_false:N \g_@@_empty_cell_bool
959 \bool_gset_false:N \g_@@_not_empty_cell_bool
960 }

```

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

```

961 \cs_new_protected:Npn \@@_node_for_cell:
962 {
963   \pgfpicture
964   \pgfsetbaseline \c_zero_dim
965   \pgfrememberpicturepositiononpagetrue
966   \pgfset
967   {
968     inner~sep = \c_zero_dim ,
969     minimum~width = \c_zero_dim
970   }
971   \pgfnode
972   { rectangle }
973   { base }
974   { \box_use_drop:N \l_@@_cell_box }
975   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
976   { }
977   \str_if_empty:NF \l_@@_name_str
978   {
979     \pgfnodealias
980     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }

```

```

981     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
982   }
983   \endpgfpicture
984 }

```

As its name says, the following command is a patch for the command `\@@_node_for_cell:`. This patch will be appended on the left of `\@@_node_for_cell:` when the construction of the cell nodes (of the form $(i-j)$) in the `\CodeBefore` is required.

```

985 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
986 {
987   \cs_new_protected:Npn \@@_patch_node_for_cell:
988   {
989     \hbox_set:Nn \l_@@_cell_box
990     {
991       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
992       \hbox_overlap_left:n
993       {
994         \pgfsys@markposition
995         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

I don't know why the following adjustment is needed when the compilation is done with XeLaTeX or with the classical way `latex`, `divps`, `ps2pdf` (or Adobe Distiller). However, it seems to work.

```

996         #1
997       }
998       \box_use:N \l_@@_cell_box
999       \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1000       \hbox_overlap_left:n
1001       {
1002         \pgfsys@markposition
1003         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1004         #1
1005       }
1006     }
1007   }
1008 }

1009 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1010 {
1011   \@@_patch_node_for_cell:n
1012   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1013 }
1014 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

1015 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3

```

```

1016 {
1017   \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1018   { g_@@_ #2 _ lines _ tl }
1019   {
1020     \use:c { @@ _ draw _ #2 : nnn }
1021     { \int_use:N \c@iRow }
1022     { \int_use:N \c@jCol }
1023     { \exp_not:n { #3 } }
1024   }
1025 }

```

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`.

```

1026 \cs_new_protected:Npn \@@_revtex_array:
1027 {
1028   \cs_set_eq:NN \@acol1 \@arrayacol
1029   \cs_set_eq:NN \@acolr \@arrayacol
1030   \cs_set_eq:NN \@acol \@arrayacol
1031   \cs_set_nopar:Npn \@halignto { }
1032   \@array@array
1033 }
1034 \cs_new_protected:Npn \@@_array:
1035 {
1036   \bool_if:NTF \c_@@_revtex_bool
1037   \@@_revtex_array:
1038   {
1039     \bool_if:NTF \l_@@_NiceTabular_bool
1040     { \dim_set_eq:NN \col@sep \tabcolsep }
1041     { \dim_set_eq:NN \col@sep \arraycolsep }
1042     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1043     { \cs_set_nopar:Npn \@halignto { } }
1044     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1045   \@tabarray
1046 }

```

`\l_@@_baseline_tl` may have the value `t`, `c` or `b`. However, if the value is `b`, we compose the `\array` (of `array`) with the option `t` and the right translation will be done further. Remark that `\str_if_eq:VnTF` is fully expandable and you need something fully expandable here.

```

1047   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1048 }

```

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.

```

1049 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1050 \cs_new_protected:Npn \@@_create_row_node:
1051 {

```

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

```

1052   \hbox
1053   {
1054     \bool_if:NT \l_@@_code_before_bool
1055     {
1056       \vtop
1057       {
1058         \skip_vertical:N 0.5\arrayrulewidth
1059         \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
1060         \skip_vertical:N -0.5\arrayrulewidth
1061       }

```

```

1062     }
1063     \pgfpicture
1064     \pgfrememberpicturepositiononpagetrue
1065     \pgfcoordinate { \l_@@_env: - row - \l_@@_succ:n \c@iRow }
1066     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1067     \str_if_empty:NF \l_@@_name_str
1068     {
1069         \pgfnodealias
1070         { \l_@@_name_str - row - \int_use:N \c@iRow }
1071         { \l_@@_env: - row - \int_use:N \c@iRow }
1072     }
1073     \endpgfpicture
1074 }
1075 }

```

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

```

1076 \cs_new:Npn \l_@@_everycr: { \noalign { \l_@@_everycr_i: } }
1077 \cs_new_protected:Npn \l_@@_everycr_i:
1078 {
1079     \int_gzero:N \c@jCol
1080     \bool_gset_false:N \g_@@_after_col_zero_bool
1081     \bool_if:NF \g_@@_row_of_col_done_bool
1082     {
1083         \l_@@_create_row_node:

```

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

```

1084     \tl_if_empty:NF \l_@@_hlines_clist
1085     {
1086         \tl_if_eq:NnF \l_@@_hlines_clist { all }
1087         {
1088             \exp_args:NNx
1089             \clist_if_in:NnT
1090             \l_@@_hlines_clist
1091             { \l_@@_succ:n \c@iRow }
1092         }
1093     }

```

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.

```

1094     \int_compare:nNnT \c@iRow > { -1 }
1095     {
1096         \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@`.

```

1097         { \hrule height \arrayrulewidth width \c_zero_dim }
1098     }
1099 }
1100 }
1101 }
1102 }

```

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

```

1103 \cs_set_protected:Npn \l_@@_newcolumnntype #1
1104 {
1105     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1106     \peek_meaning:NTF [
1107         { \newcol@ #1 }
1108         { \newcol@ #1 [ 0 ] }
1109     }

```

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

```

1110 \cs_set_protected:Npn \@@_renew_dots:
1111 {
1112   \cs_set_eq:NN \ldots \@@_Ldots
1113   \cs_set_eq:NN \cdots \@@_Cdots
1114   \cs_set_eq:NN \vdots \@@_Vdots
1115   \cs_set_eq:NN \ddots \@@_Ddots
1116   \cs_set_eq:NN \iddots \@@_Iddots
1117   \cs_set_eq:NN \dots \@@_Ldots
1118   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1119 }

```

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

```

1120 \cs_new_protected:Npn \@@_colortbl_like:
1121 {
1122   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1123   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1124   \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1125 }

```

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

```

1126 \cs_new_protected:Npn \@@_pre_array_ii:
1127 {

```

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 ⁵³.

```

1128   \bool_if:NT \c_@@_booktabs_loaded_bool
1129     { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1130   \box_clear_new:N \l_@@_cell_box
1131   \cs_if_exist:NT \theiRow
1132     { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1133   \int_gzero_new:N \c@iRow
1134   \cs_if_exist:NT \thejCol
1135     { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1136   \int_gzero_new:N \c@jCol
1137   \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}`).

```

1138   \bool_if:NT \l_@@_small_bool
1139     {
1140       \cs_set_nopar:Npn \arraystretch { 0.47 }
1141       \dim_set:Nn \arraycolsep { 1.45 pt }
1142     }

```

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`.

```

1143   \cs_set_nopar:Npn \ialign
1144     {

```

⁵³cf. `\nicematrix@redefine@check@rerun`


```

1145 \bool_if:NTF \c_@@_colortbl_loaded_bool
1146 {
1147     \CT@everycr
1148     {
1149         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1150         \@@_everycr:
1151     }
1152 }
1153 { \everycr { \@@_everycr: } }
1154 \tabskip = \c_zero_skip

```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`⁵⁴ 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`.

```

1155 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1156 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1157 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1158 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1159 \dim_gzero_new:N \g_@@_ht_row_one_dim
1160 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1161 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1162 \dim_gzero_new:N \g_@@_ht_last_row_dim
1163 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1164 \dim_gzero_new:N \g_@@_dp_last_row_dim
1165 \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`.

```

1166 \cs_set_eq:NN \ialign \@@_old_ialign:
1167 \halign
1168 }

```

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).

```

1169 \cs_set_eq:NN \@@_old_ldots \ldots
1170 \cs_set_eq:NN \@@_old_cdots \cdots
1171 \cs_set_eq:NN \@@_old_vdots \vdots
1172 \cs_set_eq:NN \@@_old_ddots \ddots
1173 \cs_set_eq:NN \@@_old_iddots \iddots
1174 \bool_if:NTF \l_@@_standard_cline_bool
1175 { \cs_set_eq:NN \cline \@@_standard_cline }
1176 { \cs_set_eq:NN \cline \@@_cline }
1177 \cs_set_eq:NN \Ldots \@@_Ldots
1178 \cs_set_eq:NN \Cdots \@@_Cdots
1179 \cs_set_eq:NN \Vdots \@@_Vdots
1180 \cs_set_eq:NN \Ddots \@@_Ddots
1181 \cs_set_eq:NN \Iddots \@@_Iddots
1182 \cs_set_eq:NN \hdottedline \@@_hdottedline:
1183 \cs_set_eq:NN \Hline \@@_Hline:
1184 \cs_set_eq:NN \Hspace \@@_Hspace:
1185 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1186 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1187 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1188 \cs_set_eq:NN \Block \@@_Block:
1189 \cs_set_eq:NN \rotate \@@_rotate:
1190 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n

```

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

```

1191 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1192 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1193 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1194 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1195 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1196 \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).

```

1197 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1198 \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).

```

1199 \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`).

```

1200 \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.

```

1201 \int_gzero_new:N \g_@@_col_total_int
1202 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1203 \@@_renew_NC@rewrite@S:
1204 \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.

```

1205 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1206 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1207 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1208 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1209 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1210 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1211 \tl_gclear_new:N \g_nicematrix_code_before_tl
1212 }

```

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

```

1213 \cs_new_protected:Npn \@@_pre_array:
1214 {
1215   \seq_gclear:N \g_@@_submatrix_seq
1216   \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1217 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1218 {
1219   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1220   {
1221     \dim_gset:Nn \g_@@_ht_last_row_dim
1222     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1223     \dim_gset:Nn \g_@@_dp_last_row_dim
1224     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1225   }

```

```

1226     }
1227     \int_compare:nNnT \l_@@_last_row_int = { -1 }
1228     {
1229         \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.

```

1230     \str_if_empty:NTF \l_@@_name_str
1231     {
1232         \cs_if_exist:cT { @@_last_row_ \int_use:N \g_@@_env_int }
1233         {
1234             \int_set:Nn \l_@@_last_row_int
1235             { \use:c { @@_last_row_ \int_use:N \g_@@_env_int } }
1236         }
1237     }
1238     {
1239         \cs_if_exist:cT { @@_last_row_ \l_@@_name_str }
1240         {
1241             \int_set:Nn \l_@@_last_row_int
1242             { \use:c { @@_last_row_ \l_@@_name_str } }
1243         }
1244     }
1245 }

```

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).

```

1246     \int_compare:nNnT \l_@@_last_col_int = { -1 }
1247     {
1248         \str_if_empty:NTF \l_@@_name_str
1249         {
1250             \cs_if_exist:cT { @@_last_col_ \int_use:N \g_@@_env_int }
1251             {
1252                 \int_set:Nn \l_@@_last_col_int
1253                 { \use:c { @@_last_col_ \int_use:N \g_@@_env_int } }
1254             }
1255         }
1256         {
1257             \cs_if_exist:cT { @@_last_col_ \l_@@_name_str }
1258             {
1259                 \int_set:Nn \l_@@_last_col_int
1260                 { \use:c { @@_last_col_ \l_@@_name_str } }
1261             }
1262         }
1263     }

```

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

```

1264     \@@_pre_array_ii:

```

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

```

1265     \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).

```

1266     \bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:

```

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

```

1267     \@@_construct_preamble:

```

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

We compute the width of both delimiters. We remember that, when the environment `{NiceArray}` is used, it's possible to specify the delimiters in the preamble (eg `[ccc]`).

```

1268 \dim_zero_new:N \l_@@_left_delim_dim
1269 \dim_zero_new:N \l_@@_right_delim_dim
1270 \bool_if:NTF \l_@@_NiceArray_bool
1271 {
1272   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1273   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1274 }
1275 {

```

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

```

1276 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1277 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1278 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1279 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1280 }

```

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).

```

1281 \hbox_set:Nw \l_@@_the_array_box
1282 \skip_horizontal:N \l_@@_left_margin_dim
1283 \skip_horizontal:N \l_@@_extra_left_margin_dim
1284 \c_math_toggle_token
1285 \bool_if:NTF \l_@@_light_syntax_bool
1286 { \use:c { @@-light-syntax } }
1287 { \use:c { @@-normal-syntax } }
1288 }

```

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

```

1289 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1290 {
1291   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1292   \bool_set_true:N \l_@@_code_before_bool

```

We go on with `\@@_pre_array:` which will (among other) execute the `\CodeBefore` (specified in the key `code-before` or after the keyword `\CodeBefore`). By definition, the `\CodeBefore` must be executed before the body of the array...

```

1293 \@@_pre_array:
1294 }

```

The `\CodeBefore`

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

```

1295 \cs_new_protected:Npn \@@_pre_code_before:
1296 {

```

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

```

1297 \int_zero_new:N \c@iRow
1298 \int_set:Nn \c@iRow
1299 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
1300 \int_zero_new:N \c@jCol
1301 \int_set:Nn \c@jCol
1302 { \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.

```

1303 \int_compare:nNnF \l_@@_last_row_int = { -2 }
1304   { \int_decr:N \c@iRow }
1305 \int_compare:nNnF \l_@@_last_col_int = { -2 }
1306   { \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.

```

1307 \pgfsys@markposition { \@@_env: - position }
1308 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1309 \pgfpicture
1310 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1311 \int_step_inline:nnn
1312   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 1 }
1313   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 + 1 }
1314   {
1315     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1316     \pgfcoordinate { \@@_env: - row - ##1 }
1317     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1318   }

```

Now, the recreation of the col nodes.

```

1319 \int_step_inline:nnn
1320   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
1321   { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 + 1 }
1322   {
1323     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1324     \pgfcoordinate { \@@_env: - col - ##1 }
1325     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1326   }

```

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

```

1327 \@@_create_diag_nodes:

```

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

```

1328 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1329 \endpgfpicture
1330 \bool_if:NT \c_@@_tikz_loaded_bool
1331   {
1332     \tikzset
1333     {
1334       every~picture / .style =
1335       { overlay , name~prefix = \@@_env: - }
1336     }
1337   }
1338 \cs_set_eq:NN \cellcolor \@@_cellcolor
1339 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1340 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1341 \cs_set_eq:NN \rowcolor \@@_rowcolor
1342 \cs_set_eq:NN \rowcolors \@@_rowcolors
1343 \cs_set_eq:NN \arraycolor \@@_arraycolor
1344 \cs_set_eq:NN \columncolor \@@_columncolor
1345 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1346 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before

```

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

```

1347 \seq_if_exist:cT
1348   { c_@@_corners_cells_ \int_use:N \g_@@_env_int _ seq }

```

```

1349     {
1350       \seq_set_eq:Nc \l_@@_corners_cells_seq
1351       { c_@@_corners_cells_ \int_use:N \g_@@_env_int _ seq }
1352     }
1353   }

```

```

1354 \cs_new_protected:Npn \@@_exec_code_before:
1355 {
1356   \seq_gclear_new:N \g_@@_colors_seq
1357   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1358   \group_begin:

```

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

```

1359   \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token

```

Here is the `\CodeBefore`. The construction is a bit complicated because `\l_@@_code_before_tl` may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do *not* execute the rest of `\l_@@_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That's why we begin with a `\q_stop`: it will be used to discard the rest of `\l_@@_code_before_tl`.

```

1360   \exp_last_unbraced:NV \@@_CodeBefore_keys: \l_@@_code_before_tl \q_stop

```

Now, all the cells which are specified to be colored by instructions in the `\CodeBefore` will actually be colored. It's a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

```

1361   \@@_actually_color:
1362   \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1363   \group_end:
1364   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1365   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1366 }

```

```

1367 \keys_define:nn { NiceMatrix / CodeBefore }
1368 {
1369   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1370   create-cell-nodes .default:n = true ,
1371   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1372   sub-matrix .value_required:n = true ,
1373   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1374   delimiters / color .value_required:n = true ,
1375   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1376 }
1377 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1378 {
1379   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1380   \@@_CodeBefore:w
1381 }

```

We have extracted the options of the keyword `\CodeBefore` in order to see whether the key `create-cell-nodes` has been used. Now, you can execute the rest of the `\CodeAfter`, excepted, of course, if we are in the first compilation.

```

1382 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1383 {
1384   \seq_if_exist:cT { @@_size_ \int_use:N \g_@@_env_int _ seq }
1385   {
1386     \@@_pre_code_before:
1387     #1
1388   }
1389 }

```

By default, if the user uses the `\CodeBefore`, only the `col` nodes, `row` nodes and `diag` nodes are available in that `\CodeBefore`. With the key `create-cell-nodes`, the cell nodes, that is to say the nodes of the form $(i-j)$ (but not the extra nodes) are also available because those nodes also are recreated and that recreation is done by the following command.

```

1390 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1391 {
1392   \int_step_inline:nnn
1393     { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 1 }
1394     { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
1395     {
1396       \int_step_inline:nnn
1397         { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
1398         { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 }
1399         {
1400           \cs_if_exist:cT
1401             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1402             {
1403               \pgfsys@getposition
1404                 { \@@_env: - ##1 - ####1 - NW }
1405               \@@_node_position:
1406               \pgfsys@getposition
1407                 { \@@_env: - ##1 - ####1 - SE }
1408               \@@_node_position_i:
1409               \@@_pgf_rect_node:nnn
1410                 { \@@_env: - ##1 - ####1 }
1411                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1412                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1413             }
1414         }
1415     }
1416 }

```

The environment `{NiceArrayWithDelims}`

```

1417 \NewDocumentEnvironment { NiceArrayWithDelims }
1418 { m m O { } m ! O { } t \CodeBefore }
1419 {
1420   \@@_provide_pgfsyspdfmark:
1421   \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.

```

1422   \bgroup

1423   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1424   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1425   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1426   \int_gzero:N \g_@@_block_box_int
1427   \dim_zero:N \g_@@_width_last_col_dim
1428   \dim_zero:N \g_@@_width_first_col_dim
1429   \bool_gset_false:N \g_@@_row_of_col_done_bool
1430   \str_if_empty:NT \g_@@_name_env_str
1431     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1432   \@@_adapt_S_column:
1433   \bool_if:NTF \l_@@_NiceTabular_bool
1434     \mode_leave_vertical:
1435     \@@_test_if_math_mode:
1436   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1437   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁵⁵. 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.

```
1438 \cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@
```

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

```
1439 \cs_if_exist:NT \tikz@library@external@loaded
1440 {
1441   \tikzexternaldisable
1442   \cs_if_exist:NT \ifstandalone
1443   { \tikzset { external / optimize = false } }
1444 }
```

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

```
1445 \int_gincr:N \g_@@_env_int
1446 \bool_if:NF \l_@@_block_auto_columns_width_bool
1447 { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

The sequence `\g_@@_blocks_seq` will contain the characteristics 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.

```
1448 \seq_gclear:N \g_@@_blocks_seq
1449 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

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

```
1450 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1451 \seq_gclear:N \g_@@_pos_of_xdots_seq
1452 \tl_if_exist:cT { g_@@_code_before_ \int_use:N \g_@@_env_int _ t1 }
1453 {
1454   \bool_set_true:N \l_@@_code_before_bool
1455   \exp_args:NNv \tl_put_right:Nn \l_@@_code_before_t1
1456   { g_@@_code_before_ \int_use:N \g_@@_env_int _ t1 }
1457 }
```

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`.

```
1458 \bool_if:NTF \l_@@_NiceArray_bool
1459 { \keys_set:nn { NiceMatrix / NiceArray } }
1460 { \keys_set:nn { NiceMatrix / pNiceArray } }
1461 { #3 , #5 }

1462 \tl_if_empty:NF \l_@@_rules_color_t1
1463 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_t1 \q_stop }
1464 % \bigskip
```

The argument `#6` is the last argument of `{NiceArrayWithDelims}`. With that argument of type “`t \CodeBefore`”, we test whether there is the keyword `\CodeBefore` at the beginning of the environment. If that keyword is present, we have now to extract all the content between that keyword `\CodeBefore` and the (other) keyword `\Body`. It's the job that will do the command `\@@_pre_array_i:w`. After that job, the command `\@@_pre_array_i:w` will go on with `\@@_pre_array:`.

```
1465 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
1466 }
1467 {
1468   \bool_if:NTF \l_@@_light_syntax_bool
1469   { \use:c { end @@-light-syntax } }
```

⁵⁵e.g. `\color[rgb]{0.5,0.5,0}`


```

1470     { \use:c { end @@-normal-syntax } }
1471     \c_math_toggle_token
1472     \skip_horizontal:N \l_@@_right_margin_dim
1473     \skip_horizontal:N \l_@@_extra_right_margin_dim
1474     \hbox_set_end:

```

End of the construction of the array (in the box `\l_@@_the_array_box`).

It 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).

```

1475     \int_compare:nNnT \l_@@_last_row_int > { -2 }
1476     {
1477         \bool_if:NF \l_@@_last_row_without_value_bool
1478         {
1479             \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1480             {
1481                 \@@_error:n { Wrong~last~row }
1482                 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1483             }
1484         }
1485     }

```

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”.⁵⁶

```

1486     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1487     \bool_if:nTF \g_@@_last_col_found_bool
1488     { \int_gdecr:N \c@jCol }
1489     {
1490         \int_compare:nNnT \l_@@_last_col_int > { -1 }
1491         { \@@_error:n { last~col~not~used } }
1492     }

```

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

```

1493     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1494     \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. 107).

```

1495     \int_compare:nNnT \l_@@_first_col_int = 0
1496     {
1497         \skip_horizontal:N \col@sep
1498         \skip_horizontal:N \g_@@_width_first_col_dim
1499     }

```

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

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

```

1500     \bool_if:NTF \l_@@_NiceArray_bool
1501     {
1502         \str_case:VnF \l_@@_baseline_tl
1503         {
1504             b \@@_use_arraybox_with_notes_b:
1505             c \@@_use_arraybox_with_notes_c:
1506         }
1507         \@@_use_arraybox_with_notes:
1508     }

```

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).

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

```

1509 {
1510   \int_compare:nNnTF \l_@@_first_row_int = 0
1511   {
1512     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1513     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1514   }
1515   { \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”.⁵⁷

```

1516   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1517   {
1518     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1519     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1520   }
1521   { \dim_zero:N \l_tmpb_dim }
1522   \hbox_set:Nn \l_tmpa_box
1523   {
1524     \c_math_toggle_token
1525     \tl_if_empty:NF \l_@@_delimiters_color_tl
1526     { \color { \l_@@_delimiters_color_tl } }
1527     \exp_after:wN \left \g_@@_left_delim_tl
1528     \vcenter
1529     {

```

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.

```

1530       \skip_vertical:N -\l_tmpa_dim
1531       \hbox
1532       {
1533         \bool_if:NTF \l_@@_NiceTabular_bool
1534         { \skip_horizontal:N -\tabcolsep }
1535         { \skip_horizontal:N -\arraycolsep }
1536         \@@_use_arraybox_with_notes_c:
1537         \bool_if:NTF \l_@@_NiceTabular_bool
1538         { \skip_horizontal:N -\tabcolsep }
1539         { \skip_horizontal:N -\arraycolsep }
1540       }

```

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

```

1541       \skip_vertical:N -\l_tmpb_dim
1542     }

```

Curiously, we have to put again the following specification of color. Otherwise, with XeLaTeX (and not with the other engines), the closing delimiter is not colored.

```

1543     \tl_if_empty:NF \l_@@_delimiters_color_tl
1544     { \color { \l_@@_delimiters_color_tl } }
1545     \exp_after:wN \right \g_@@_right_delim_tl
1546     \c_math_toggle_token
1547   }

```

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

We will put the box in the TeX flow. However, we have a small work to do when the option `delimiters/max-width` is used.

```

1548     \bool_if:NTF \l_@@_delimiters_max_width_bool
1549     {
1550       \@@_put_box_in_flow_bis:nn
1551       \g_@@_left_delim_tl \g_@@_right_delim_tl
1552     }
1553     \@@_put_box_in_flow:
1554   }

```

⁵⁷A 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 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. 108).

```

1555 \bool_if:NT \g_@@_last_col_found_bool
1556 {
1557     \skip_horizontal:N \g_@@_width_last_col_dim
1558     \skip_horizontal:N \col@sep
1559 }
1560 \bool_if:NF \l_@@_Matrix_bool
1561 {
1562     \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1563     { \@@_error:n { columns-not-used } }
1564 }
1565 \group_begin:
1566 \globaldefs = 1
1567 \@@_msg_redirect_name:nn { columns-not-used } { error }
1568 \group_end:
1569 \@@_after_array:

```

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

```

1570 \egroup
1571 \bool_if:NT \c_@@_footnote_bool \endsavenotes
1572 }

```

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

We construct the preamble of the array

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

The preamble given by the final user is in `\g_@@_preamble_tl` and the modified version will be stored in `\g_@@_preamble_tl` also.

```

1573 \cs_new_protected:Npn \@@_construct_preamble:
1574 {

```

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

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

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

```

1575 \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).

```

1576 \bool_if:NF \l_@@_Matrix_bool
1577 {
1578     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1579     \@@_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`).

```

1580 \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1581 \@tempwatrue

```

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

```
1582 \whilesw \if@tempswa \fi { \@tempwafalse \the \NC@list }
```

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

```
1583 \int_gzero_new:N \c@jCol
1584 \tl_gclear:N \g_@@_preamble_tl
1585 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1586 {
1587   \tl_gset:Nn \g_@@_preamble_tl
1588   { ! { \skip_horizontal:N \arrayrulewidth } }
1589 }
1590 {
1591   \clist_if_in:NnT \l_@@_vlines_clist 1
1592   {
1593     \tl_gset:Nn \g_@@_preamble_tl
1594     { ! { \skip_horizontal:N \arrayrulewidth } }
1595   }
1596 }
```

The sequence `\g_@@_cols_vlsim_seq` will contain the numbers of the columns where you will to have to draw vertical lines in the potential sub-matrices (hence the name `vlsim`).

```
1597 \seq_clear:N \g_@@_cols_vlsim_seq
```

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

```
1598 \int_zero:N \l_tmpa_int
```

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

```
1599 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1600 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1601 }
```

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

```
1602 \bool_if:NT \l_@@_colortbl_like_bool
1603 {
1604   \regex_replace_all:NnN
1605   \c_@@_columncolor_regex
1606   { \c { @@_columncolor_preamble } }
1607   \g_@@_preamble_tl
1608 }
```

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

```
1609 \group_end:
```

If there was delimiters at the beginning or at the end of the preamble, the environment `{NiceArray}` is transformed into an environment `{xNiceMatrix}`.

```
1610 \bool_lazy_or:nnT
1611 { ! \str_if_eq_p:Nn \g_@@_left_delim_tl { . } }
1612 { ! \str_if_eq_p:Nn \g_@@_right_delim_tl { . } }
1613 { \bool_set_false:N \l_@@_NiceArray_bool }
```

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

```
1614 \int_compare:nNnTF \l_@@_first_col_int = 0
1615 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1616 {
1617   \bool_lazy_all:nT
1618   {
1619     \l_@@_NiceArray_bool
```

```

1620         { \bool_not_p:n \l_@@_NiceTabular_bool }
1621         { \tl_if_empty_p:N \l_@@_vlines_clist }
1622         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1623     }
1624     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1625 }
1626 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1627 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1628 {
1629     \bool_lazy_all:nT
1630     {
1631         \l_@@_NiceArray_bool
1632         { \bool_not_p:n \l_@@_NiceTabular_bool }
1633         { \tl_if_empty_p:N \l_@@_vlines_clist }
1634         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1635     }
1636     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1637 }

```

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`).

```

1638     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1639     {
1640         \tl_gput_right:Nn \g_@@_preamble_tl
1641         { > { \@@_error_too_much_cols: } 1 }
1642     }
1643 }

1644 \cs_new_protected:Npn \@@_patch_preamble:n #1
1645 {
1646     \str_case:nnF { #1 }
1647     {
1648         c { \@@_patch_preamble_i:n #1 }
1649         l { \@@_patch_preamble_i:n #1 }
1650         r { \@@_patch_preamble_i:n #1 }
1651         > { \@@_patch_preamble_ii:nn #1 }
1652         ! { \@@_patch_preamble_ii:nn #1 }
1653         @ { \@@_patch_preamble_ii:nn #1 }
1654         | { \@@_patch_preamble_iii:n #1 }
1655         p { \@@_patch_preamble_iv:nnn t #1 }
1656         m { \@@_patch_preamble_iv:nnn c #1 }
1657         b { \@@_patch_preamble_iv:nnn b #1 }
1658         \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
1659         \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1660         \@@_true_c: { \@@_patch_preamble_vi:n #1 }
1661         ( { \@@_patch_preamble_vii:nn #1 }
1662         [ { \@@_patch_preamble_vii:nn #1 }
1663         \{ { \@@_patch_preamble_vii:nn #1 }
1664         ) { \@@_patch_preamble_viii:nn #1 }
1665         ] { \@@_patch_preamble_viii:nn #1 }
1666         \} { \@@_patch_preamble_viii:nn #1 }
1667         C { \@@_error:nn { old~column~type } #1 }
1668         L { \@@_error:nn { old~column~type } #1 }
1669         R { \@@_error:nn { old~column~type } #1 }
1670         \q_stop { }
1671     }
1672 }
1673 \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1674 { \@@_patch_preamble_xi:n #1 }
1675 {
1676     \str_if_eq:VnTF \l_@@_letter_vlism_tl { #1 }
1677     {

```

```

1678         \seq_gput_right:Nx \g_@@_cols_vlism_seq
1679         { \int_eval:n { \c@jCol + 1 } }
1680     \tl_gput_right:Nx \g_@@_preamble_tl
1681     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1682     \@@_patch_preamble:n
1683 }
1684 {
1685     \bool_lazy_and:nnTF
1686     { \str_if_eq_p:nn { : } { #1 } }
1687     \c_@@_arydshln_loaded_bool
1688     {
1689         \tl_gput_right:Nn \g_@@_preamble_tl { : }
1690         \@@_patch_preamble:n
1691     }
1692     { \@@_fatal:nn { unknown~column~type } { #1 } }
1693 }
1694 }
1695 }
1696 }

```

For c, l and r

```

1697 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1698 {
1699     \tl_gput_right:Nn \g_@@_preamble_tl
1700     {
1701         > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } }
1702         #1
1703         < \@@_end_Cell:
1704     }

```

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

```

1705     \int_gincr:N \c@jCol
1706     \@@_patch_preamble_x:n
1707 }

```

For >, ! and @

```

1708 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1709 {
1710     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1711     \@@_patch_preamble:n
1712 }

```

For |

```

1713 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1714 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1715     \int_incr:N \l_tmpa_int
1716     \@@_patch_preamble_iii_i:n
1717 }
1718 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1719 {
1720     \str_if_eq:nnTF { #1 } |
1721     { \@@_patch_preamble_iii:n | }
1722     {
1723         \tl_gput_right:Nx \g_@@_preamble_tl
1724         {
1725             \exp_not:N !
1726             {
1727                 \skip_horizontal:n
1728                 {
1729                     \dim_eval:n
1730                     {

```

```

1731         \arrayrulewidth * \l_tmpa_int
1732         + \doublerulesep * ( \l_tmpa_int - 1)
1733     }
1734 }
1735 }
1736 }
1737 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1738 { \@@_vline:nn { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } }
1739 \int_zero:N \l_tmpa_int
1740 \@@_patch_preamble:n #1
1741 }
1742 }

```

For p, m and b

```

1743 \cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
1744 {
1745     \tl_gput_right:Nn \g_@@_preamble_tl
1746     {
1747         > {
1748             \@@_Cell:
1749             \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
1750             \mode_leave_vertical:
1751             \arraybackslash
1752             \vrule height \box_ht:N \@@_arstrutbox depth 0 pt width 0 pt
1753         }
1754         c
1755         < {
1756             \vrule height 0 pt depth \box_dp:N \@@_arstrutbox width 0 pt
1757             \end { minipage }
1758             \@@_end_Cell:
1759         }
1760     }

```

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

```

1761     \int_gincr:N \c@jCol
1762     \@@_patch_preamble_x:n
1763 }

```

For w and W

```

1764 \cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
1765 {
1766     \tl_gput_right:Nn \g_@@_preamble_tl
1767     {
1768         > {
1769             \hbox_set:Nw \l_@@_cell_box
1770             \@@_Cell:
1771             \tl_set:Nn \l_@@_cell_type_tl { #3 }
1772         }
1773         c
1774         < {
1775             \@@_end_Cell:
1776             #1
1777             \hbox_set_end:
1778             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1779             \@@_adjust_size_box:
1780             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
1781         }
1782     }

```

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

```

1783     \int_gincr:N \c@jCol
1784     \@@_patch_preamble_x:n
1785 }

```

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

```

1786 \cs_new_protected:Npn \@@_patch_preamble_vi:n #1
1787 {
1788   \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

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

```

1789   \int_gincr:N \c@jCol
1790   \@@_patch_preamble_x:n
1791 }

```

For (, [and \{.

```

1792 \cs_new_protected:Npn \@@_patch_preamble_vii:nn #1 #2
1793 {
1794   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }

```

If we are before the column 1 and not in {NiceArray}, we reserve space for the left delimiter.

```

1795   \int_compare:nNnTF \c@jCol = \c_zero_int
1796   {
1797     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
1798     {

```

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

```

1799       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1800       \tl_gset:Nn \g_@@_right_delim_tl { . }
1801       \@@_patch_preamble:n #2
1802     }
1803   {
1804     \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1805     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1806       { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
1807     \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
1808     {
1809       \@@_error:nn { delimiter~after~opening } { #2 }
1810       \@@_patch_preamble:n
1811     }
1812     { \@@_patch_preamble:n #2 }
1813   }
1814 }
1815 {
1816   \tl_gput_right:Nx \g_@@_internal_code_after_tl
1817     { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
1818   \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
1819   {
1820     \@@_error:nn { delimiter~after~opening } { #2 }
1821     \@@_patch_preamble:n
1822   }
1823   { \@@_patch_preamble:n #2 }
1824 }
1825 }

```

For),] and \}. We have two arguments for the following command because we directly read the following letter in the preamble (we have to see whether we have a opening delimiter following and we also have to see whether we are at the end of the preamble because, in that case, our letter must be considered as the right delimiter of the environment if the environment is {NiceArray}).

```

1826 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
1827 {
1828   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
1829   \tl_if_in:nnTF { ) ] \} } { #2 }
1830   { \@@_patch_preamble_viii_i:nnn #1 #2 }
1831   {
1832     \tl_if_eq:nnTF { \q_stop } { #2 }
1833     {

```



```

1834 \str_if_eq:VnTF \g_@@_right_delim_tl { . }
1835 { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
1836 {
1837   \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1838   \tl_gput_right:Nx \g_@@_internal_code_after_tl
1839   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1840   \@@_patch_preamble:n #2
1841 }
1842 }
1843 {
1844   \tl_if_in:nnT { ( [ \{ } { #2 }
1845   { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
1846   \tl_gput_right:Nx \g_@@_internal_code_after_tl
1847   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1848   \@@_patch_preamble:n #2
1849 }
1850 }
1851 }
1852 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nnn #1 #2 #3
1853 {
1854   \tl_if_eq:nnTF { \q_stop } { #3 }
1855   {
1856     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
1857     {
1858       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1859       \tl_gput_right:Nx \g_@@_internal_code_after_tl
1860       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1861       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1862     }
1863     {
1864       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1865       \tl_gput_right:Nx \g_@@_internal_code_after_tl
1866       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1867       \@@_error:nn { double-closing-delimiter } { #2 }
1868     }
1869   }
1870   {
1871     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1872     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1873     \@@_error:nn { double-closing-delimiter } { #2 }
1874     \@@_patch_preamble:n #3
1875   }
1876 }
1877 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
1878 {
1879   \tl_gput_right:Nn \g_@@_preamble_tl
1880   { ! { \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`.

```

1881 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1882 { \@@_vdottedline:n { \int_use:N \c@jCol } }
1883 \@@_patch_preamble:n
1884 }

```

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.

```

1885 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
1886 {
1887   \str_if_eq:nnTF { #1 } { < }
1888   \@@_patch_preamble_ix:n
1889   {
1890     \tl_if_eq:NnTF \l_@@_vlines_clist { all }

```

```

1891     {
1892         \tl_gput_right:Nn \g_@@_preamble_tl
1893         { ! { \skip_horizontal:N \arrayrulewidth } }
1894     }
1895     {
1896         \exp_args:NNx
1897         \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
1898         {
1899             \tl_gput_right:Nn \g_@@_preamble_tl
1900             { ! { \skip_horizontal:N \arrayrulewidth } }
1901         }
1902     }
1903     \@@_patch_preamble:n { #1 }
1904 }
1905 }
1906 \cs_new_protected:Npn \@@_patch_preamble_ix:n #1
1907 {
1908     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
1909     \@@_patch_preamble_x:n
1910 }

```

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`).

```

1911 \cs_new_protected:Npn \@@_put_box_in_flow:
1912 {
1913     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
1914     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
1915     \tl_if_eq:NnTF \l_@@_baseline_tl { c }
1916     { \box_use_drop:N \l_tmpa_box }
1917     \@@_put_box_in_flow_i:
1918 }

```

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

```

1919 \cs_new_protected:Npn \@@_put_box_in_flow_i:
1920 {
1921     \pgfpicture
1922     \@@_qpoint:n { row - 1 }
1923     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1924     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
1925     \dim_gadd:Nn \g_tmpa_dim \pgf@y
1926     \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).

```

1927     \str_if_in:NnTF \l_@@_baseline_tl { line- }
1928     {
1929         \int_set:Nn \l_tmpa_int
1930         {
1931             \str_range:Nnn
1932             \l_@@_baseline_tl
1933             6
1934             { \tl_count:V \l_@@_baseline_tl }
1935         }
1936         \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
1937     }
1938     {
1939         \str_case:VnF \l_@@_baseline_tl
1940         {
1941             { t } { \int_set:Nn \l_tmpa_int 1 }

```

```

1942         { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
1943     }
1944     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
1945     \bool_lazy_or:nnT
1946     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
1947     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
1948     {
1949         \@@_error:n { bad-value-for-baseline }
1950         \int_set:Nn \l_tmpa_int 1
1951     }
1952     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

1953     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
1954 }
1955 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

1956 \endpgfpicture
1957 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
1958 \box_use_drop:N \l_tmpa_box
1959 }

```

The following command is *always* used by `{NiceArrayWithDelims}` (even if, in fact, there is no tabular notes: in fact, it's not possible to know whether there is tabular notes or not before the composition of the blocks).

```

1960 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
1961 {

```

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

```

1962     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
1963     \box_use_drop:N \l_@@_the_array_box

```

We have to draw the blocks right now because there may be tabular notes in some blocks (which are not mono-column: the blocks which are mono-column have been composed in boxes yet)... and we have to create (potentially) the extra nodes before creating the blocks since there are `medium` nodes to create for the blocks.

```

1964     \@@_create_extra_nodes:
1965     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
1966     \bool_lazy_or:nnT
1967     { \int_compare_p:nNn \c@tabularnote > 0 }
1968     { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
1969     \@@_insert_tabularnotes:
1970     \end { minipage }
1971 }

```

```

1972 \cs_new_protected:Npn \@@_insert_tabularnotes:
1973 {
1974     \skip_vertical:N 0.65ex

```

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

```

1975     \group_begin:
1976     \l_@@_notes_code_before_tl
1977     \tl_if_empty:NF \l_@@_tabularnote_tl { \l_@@_tabularnote_tl \par }

```

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

```

1978     \int_compare:nNnT \c@tabularnote > 0
1979     {
1980         \bool_if:NTF \l_@@_notes_para_bool
1981         {
1982             \begin { tabularnotes* }
1983             \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
1984             \end { tabularnotes* }

```

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

```

1985         \par
1986     }
1987     {
1988         \tabularnotes
1989         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
1990     \endtabularnotes
1991     }
1992 }
1993 \unskip
1994 \group_end:
1995 \bool_if:NT \l_@@_notes_bottomrule_bool
1996 {
1997     \bool_if:NTF \c_@@_booktabs_loaded_bool
1998     {

```

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

```

1999         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2000     { \CT@arc@ \hrule height \heavyrulewidth }
2001     }
2002     { \@@_error:n { bottomrule~without~booktabs } }
2003     }
2004 \l_@@_notes_code_after_tl
2005 \seq_gclear:N \g_@@_tabularnotes_seq
2006 \int_gzero:N \c@tabularnote
2007 }

```

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`.

```

2008 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2009 {
2010     \pgfpicture
2011     \@@_qpoint:n { row - 1 }
2012     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2013     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2014     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2015     \endpgfpicture
2016     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2017     \int_compare:nNnT \l_@@_first_row_int = 0
2018     {
2019         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2020         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2021     }
2022     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2023 }

```

Now, the general case.

```

2024 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2025 {

```

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

```

2026     \tl_if_eq:NnT \l_@@_baseline_tl { t }
2027     { \tl_set:Nn \l_@@_baseline_tl { 1 } }

```

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

```

2028     \pgfpicture
2029     \@@_qpoint:n { row - 1 }
2030     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2031     \str_if_in:NnTF \l_@@_baseline_tl { line- }

```

```

2032 {
2033   \int_set:Nn \l_tmpa_int
2034   {
2035     \str_range:Nnn
2036       \l_@@_baseline_tl
2037       6
2038       { \tl_count:V \l_@@_baseline_tl }
2039   }
2040   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2041 }
2042 {
2043   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2044   \bool_lazy_or:nnT
2045     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2046     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2047   {
2048     \@@_error:n { bad~value~for~baseline }
2049     \int_set:Nn \l_tmpa_int 1
2050   }
2051   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2052 }
2053 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2054 \endpgfpicture
2055 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2056 \int_compare:nNnT \l_@@_first_row_int = 0
2057 {
2058   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2059   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2060 }
2061 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2062 }

```

The command `\@@_put_box_in_flow_bis:` is used when the option `delimiters/max-width` is used because, in this case, we have to adjust the widths of the delimiters. The arguments `#1` and `#2` are the delimiters specified by the user.

```

2063 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2064 {

```

We will compute the real width of both delimiters used.

```

2065   \dim_zero_new:N \l_@@_real_left_delim_dim
2066   \dim_zero_new:N \l_@@_real_right_delim_dim
2067   \hbox_set:Nn \l_tmpb_box
2068   {
2069     \c_math_toggle_token
2070     \left #1
2071     \vcenter
2072     {
2073       \vbox_to_ht:nn
2074         { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2075         { }
2076     }
2077     \right .
2078     \c_math_toggle_token
2079   }
2080   \dim_set:Nn \l_@@_real_left_delim_dim
2081   { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
2082   \hbox_set:Nn \l_tmpb_box
2083   {
2084     \c_math_toggle_token
2085     \left .
2086     \vbox_to_ht:nn
2087       { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2088       { }

```

```

2089     \right #2
2090     \c_math_toggle_token
2091   }
2092   \dim_set:Nn \l_@@_real_right_delim_dim
2093     { \box_wd:N \l_tmpb_box - \nullldelimiterspace }

```

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

```

2094   \skip_horizontal:N \l_@@_left_delim_dim
2095   \skip_horizontal:N -\l_@@_real_left_delim_dim
2096   \@@_put_box_in_flow:
2097   \skip_horizontal:N \l_@@_right_delim_dim
2098   \skip_horizontal:N -\l_@@_real_right_delim_dim
2099 }

```

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).

```

2100 \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).

```

2101 {
2102   \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`).

```

2103   { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2104 }
2105 {
2106   \@@_create_col_nodes:
2107   \endarray
2108 }

```

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`).

```

2109 \NewDocumentEnvironment { @@-light-syntax } { b }
2110 {

```

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`.

```

2111   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
2112   \tl_map_inline:nn { #1 }
2113   {
2114     \str_if_eq:nnT { ##1 } { & }
2115     { \@@_fatal:n { ampersand~in~light-syntax } }
2116     \str_if_eq:nnT { ##1 } { \ }
2117     { \@@_fatal:n { double-backslash~in~light-syntax } }
2118   }

```

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

```

2119   \@@_light_syntax_i #1 \CodeAfter \q_stop
2120 }

```

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`.

```

2121 { }
2122 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2123 {
2124   \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).

```

2125   \seq_gclear_new:N \g_@@_rows_seq
2126   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2127   \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.

```

2128   \int_compare:nNnT \l_@@_last_row_int = { -1 }
2129     { \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`).

```

2130   \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.

```

2131   \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2132   \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
2133   \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2134   \@@_create_col_nodes:
2135   \endarray
2136 }
2137 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2138 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
2139 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2140 {
2141   \seq_gclear_new:N \g_@@_cells_seq
2142   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2143   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2144   \l_tmpa_tl
2145   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2146 }

```

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).

```

2147 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2148 {
2149   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2150   { \@@_fatal:n { empty~environment } }

```

We reprint in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```

2151   \end { #2 }
2152 }

```

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).

```

2153 \cs_new:Npn \@@_create_col_nodes:
2154 {
2155   \crrc
2156   \int_compare:nNnT \l_@@_first_col_int = 0
2157   {
2158     \omit

```

```

2159 \hbox_overlap_left:n
2160 {
2161   \bool_if:NT \l_@@_code_before_bool
2162     { \pgfsys@markposition { \@@_env: - col - 0 } }
2163   \pgfpicture
2164     \pgfrememberpicturerepositiononpagetrue
2165     \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
2166     \str_if_empty:NF \l_@@_name_str
2167       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2168     \endpgfpicture
2169     \skip_horizontal:N 2\col@sep
2170     \skip_horizontal:N \g_@@_width_first_col_dim
2171   }
2172   &
2173 }
2174 \omit

```

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

```

2175 \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`).

```

2176 \int_compare:nNnTF \l_@@_first_col_int = 0
2177 {
2178   \bool_if:NT \l_@@_code_before_bool
2179   {
2180     \hbox
2181     {
2182       \skip_horizontal:N -0.5\arrayrulewidth
2183       \pgfsys@markposition { \@@_env: - col - 1 }
2184       \skip_horizontal:N 0.5\arrayrulewidth
2185     }
2186   }
2187   \pgfpicture
2188   \pgfrememberpicturerepositiononpagetrue
2189   \pgfcoordinate { \@@_env: - col - 1 }
2190     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
2191   \str_if_empty:NF \l_@@_name_str
2192     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2193   \endpgfpicture
2194 }
2195 {
2196   \bool_if:NT \l_@@_code_before_bool
2197   {
2198     \hbox
2199     {
2200       \skip_horizontal:N 0.5\arrayrulewidth
2201       \pgfsys@markposition { \@@_env: - col - 1 }
2202       \skip_horizontal:N -0.5\arrayrulewidth
2203     }
2204   }
2205   \pgfpicture
2206   \pgfrememberpicturerepositiononpagetrue
2207   \pgfcoordinate { \@@_env: - col - 1 }
2208     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
2209   \str_if_empty:NF \l_@@_name_str
2210     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2211   \endpgfpicture
2212 }

```

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.

```

2213 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2214 \bool_if:NF \l_@@_auto_columns_width_bool
2215 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2216 {
2217   \bool_lazy_and:nnTF
2218     \l_@@_auto_columns_width_bool
2219     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2220     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2221     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2222     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2223 }
2224 \skip_horizontal:N \g_tmpa_skip
2225 \hbox
2226 {
2227   \bool_if:NT \l_@@_code_before_bool
2228   {
2229     \hbox
2230     {
2231       \skip_horizontal:N -0.5\arrayrulewidth
2232       \pgfsys@markposition { \@@_env: - col - 2 }
2233       \skip_horizontal:N 0.5\arrayrulewidth
2234     }
2235   }
2236   \pgfpicture
2237   \pgfrememberpicturerepositiononpagetrue
2238   \pgfcoordinate { \@@_env: - col - 2 }
2239   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2240   \str_if_empty:NF \l_@@_name_str
2241   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2242   \endpgfpicture
2243 }

```

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.

```

2244 \int_gset:Nn \g_tmpa_int 1
2245 \bool_if:NTF \g_@@_last_col_found_bool
2246 { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
2247 { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
2248 {
2249   &
2250   \omit
2251   \int_gincr:N \g_tmpa_int

```

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

```

2252 \skip_horizontal:N \g_tmpa_skip
2253 \bool_if:NT \l_@@_code_before_bool
2254 {
2255   \hbox
2256   {
2257     \skip_horizontal:N -0.5\arrayrulewidth
2258     \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2259     \skip_horizontal:N 0.5\arrayrulewidth
2260   }
2261 }

```

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

```

2262 \pgfpicture
2263 \pgfrememberpicturerepositiononpagetrue
2264 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2265 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2266 \str_if_empty:NF \l_@@_name_str
2267 {

```

```

2268         \pgfnodealias
2269         { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2270         { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2271     }
2272     \endpgfpicture
2273 }
2274 \bool_if:NT \g_@@_last_col_found_bool
2275 {
2276     \hbox_overlap_right:n
2277     {
2278         % \skip_horizontal:N \col@sep
2279         \skip_horizontal:N \g_@@_width_last_col_dim
2280         \bool_if:NT \l_@@_code_before_bool
2281         {
2282             \pgfsys@markposition
2283             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2284         }
2285         \pgfpicture
2286         \pgfrememberpicturepositiononpagetrue
2287         \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2288         \pgfpointorigin
2289         \str_if_empty:NF \l_@@_name_str
2290         {
2291             \pgfnodealias
2292             { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2293             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2294         }
2295         \endpgfpicture
2296     }
2297 }
2298 \cr
2299 }

```

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

```

2300 \tl_const:Nn \c_@@_preamble_first_col_tl
2301 {
2302     >
2303     {

```

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

```

2304     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
2305     \bool_gset_true:N \g_@@_after_col_zero_bool
2306     \@@_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.

```

2307     \hbox_set:Nw \l_@@_cell_box
2308     \@@_math_toggle_token:
2309     \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”.

```

2310     \bool_lazy_and:nnT
2311     { \int_compare_p:nNn \c@iRow > 0 }
2312     {
2313         \bool_lazy_or_p:nn
2314         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2315         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2316     }
2317     {
2318         \l_@@_code_for_first_col_tl
2319         \xglobal \colorlet { nicematrix-first-col } { . }

```

```

2320     }
2321 }

```

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`.

```

2322   l
2323   <
2324   {
2325     \@@_math_toggle_token:
2326     \hbox_set_end:
2327     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2328     \@@_adjust_size_box:
2329     \@@_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.

```

2330     \dim_gset:Nn \g_@@_width_first_col_dim
2331     { \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.

```

2332     \hbox_overlap_left:n
2333     {
2334       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2335       \@@_node_for_cell:
2336       { \box_use_drop:N \l_@@_cell_box }
2337       \skip_horizontal:N \l_@@_left_delim_dim
2338       \skip_horizontal:N \l_@@_left_margin_dim
2339       \skip_horizontal:N \l_@@_extra_left_margin_dim
2340     }
2341     \bool_gset_false:N \g_@@_empty_cell_bool
2342     \skip_horizontal:N -2\col@sep
2343   }
2344 }

```

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

```

2345 \tl_const:Nn \c_@@_preamble_last_col_tl
2346 {
2347   >
2348   {

```

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

```

2349     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n

```

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

```

2350     \bool_gset_true:N \g_@@_last_col_found_bool
2351     \int_gincr:N \c@jCol
2352     \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.

```

2353     \hbox_set:Nw \l_@@_cell_box
2354     \@@_math_toggle_token:
2355     \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”.

```

2356     \int_compare:nNnT \c@iRow > 0
2357     {
2358       \bool_lazy_or:nnT
2359       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2360       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2361       {
2362         \l_@@_code_for_last_col_tl
2363         \xglobal \colorlet { nicematrix-last-col } { . }
2364       }

```

```

2365     }
2366   }
2367   l
2368   <
2369   {
2370     \@@_math_toggle_token:
2371     \hbox_set_end:
2372     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2373     \@@_adjust_size_box:
2374     \@@_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.

```

2375     \dim_gset:Nn \g_@@_width_last_col_dim
2376     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2377     \skip_horizontal:N -2\col@sep

```

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

```

2378     \hbox_overlap_right:n
2379     {
2380       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2381       {
2382         \skip_horizontal:N \l_@@_right_delim_dim
2383         \skip_horizontal:N \l_@@_right_margin_dim
2384         \skip_horizontal:N \l_@@_extra_right_margin_dim
2385         \@@_node_for_cell:
2386       }
2387     }
2388     \bool_gset_false:N \g_@@_empty_cell_bool
2389   }
2390 }

```

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.

```

2391 \NewDocumentEnvironment { NiceArray } { }
2392 {
2393   \bool_set_true:N \l_@@_NiceArray_bool
2394   \str_if_empty:NT \g_@@_name_env_str
2395   { \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).

```

2396   \NiceArrayWithDelims . .
2397 }
2398 { \endNiceArrayWithDelims }

```

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

```

2399 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2400 {
2401   \NewDocumentEnvironment { #1 NiceArray } { }
2402   {
2403     \str_if_empty:NT \g_@@_name_env_str
2404     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2405     \@@_test_if_math_mode:
2406     \NiceArrayWithDelims #2 #3
2407   }
2408   { \endNiceArrayWithDelims }
2409 }

```

```

2410 \@@_def_env:nnn p ( )
2411 \@@_def_env:nnn b [ ]
2412 \@@_def_env:nnn B \{ \}
2413 \@@_def_env:nnn v | |
2414 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2415 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2416 {
2417   \bool_set_true:N \l_@@_Matrix_bool
2418   \use:c { #1 NiceArray }
2419   {
2420     *
2421     {
2422       \int_compare:nNnTF \l_@@_last_col_int < 0
2423         \c@MaxMatrixCols
2424         { \@@_pred:n \l_@@_last_col_int }
2425     }
2426     { > \@@_Cell: #2 < \@@_end_Cell: }
2427   }
2428 }
2429 \clist_map_inline:nn { { } , p , b , B , v , V }
2430 {
2431   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
2432   {
2433     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2434     \tl_set:Nn \l_@@_type_of_col_tl c
2435     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2436     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2437   }
2438   { \use:c { end #1 NiceArray } }
2439 }

```

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

```

2440 \cs_new_protected:Npn \@@_NotEmpty:
2441 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

The environments {NiceTabular} and {NiceTabular*}

```

2442 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
2443 {
2444   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2445   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2446   \bool_set_true:N \l_@@_NiceTabular_bool
2447   \NiceArray { #2 }
2448 }
2449 { \endNiceArray }

2450 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
2451 {
2452   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2453   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
2454   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2455   \bool_set_true:N \l_@@_NiceTabular_bool
2456   \NiceArray { #3 }
2457 }
2458 { \endNiceArray }

```

After the construction of the array

```

2459 \cs_new_protected:Npn \@@_after_array:
2460 {
2461   \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.

```

2462   \bool_if:NT \g_@@_last_col_found_bool
2463   { \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`.

```

2464   \bool_if:NT \l_@@_last_col_without_value_bool
2465   {
2466     \dim_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int
2467     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2468     \iow_shipout:Nx \@mainaux
2469     {
2470       \cs_gset:cpn { @@_last_col_ \int_use:N \g_@@_env_int }
2471       { \int_use:N \g_@@_col_total_int }
2472     }
2473     \str_if_empty:NF \l_@@_name_str
2474     {
2475       \iow_shipout:Nx \@mainaux
2476       {
2477         \cs_gset:cpn { @@_last_col_ \l_@@_name_str }
2478         { \int_use:N \g_@@_col_total_int }
2479       }
2480     }
2481     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2482   }

```

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.

```

2483   \bool_if:NT \l_@@_last_row_without_value_bool
2484   {
2485     \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.

```

2486   \bool_if:NF \l_@@_light_syntax_bool
2487   {
2488     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2489     \iow_shipout:Nx \@mainaux
2490     {
2491       \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int }
2492       { \int_use:N \g_@@_row_total_int }
2493     }

```

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.

```

2494     \str_if_empty:NF \l_@@_name_str
2495     {
2496       \iow_shipout:Nx \@mainaux
2497       {
2498         \cs_gset:cpn { @@_last_row_ \l_@@_name_str }
2499         { \int_use:N \g_@@_row_total_int }
2500       }
2501     }
2502     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2503   }

```

```
2504 }
```

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

```
2505 \bool_if:NT \l_@@_code_before_bool
2506 {
2507   \iow_now:Nn \@mainaux \ExplSyntaxOn
2508   \iow_now:Nx \@mainaux
2509   { \seq_clear_new:c { @@_size _ \int_use:N \g_@@_env_int _ seq } }
2510   \iow_now:Nx \@mainaux
2511   {
2512     \seq_gset_from_clist:cn { @@_size _ \int_use:N \g_@@_env_int _ seq }
2513     {
2514       \int_use:N \l_@@_first_row_int ,
2515       \int_use:N \g_@@_row_total_int ,
2516       \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`.

```
2517       \bool_lazy_and:nnTF
2518       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
2519       { \bool_not_p:n \g_@@_last_col_found_bool }
2520       \@@_succ:n
2521       \int_use:N
2522       \g_@@_col_total_int
2523     }
```

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

```
2524     \seq_gset_from_clist:cn
2525     { c_@@_pos_of_blocks _ \int_use:N \g_@@_env_int _ seq }
2526     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
2527   }
2528   \iow_now:Nn \@mainaux \ExplSyntaxOff
2529 }
```

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

```
2530 \@@_create_diag_nodes:
```

By default, the diagonal lines will be parallelized⁵⁸. 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.

```
2531 \bool_if:NT \l_@@_parallelize_diags_bool
2532 {
2533   \int_gzero_new:N \g_@@_ddots_int
2534   \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.

```
2535   \dim_gzero_new:N \g_@@_delta_x_one_dim
2536   \dim_gzero_new:N \g_@@_delta_y_one_dim
2537   \dim_gzero_new:N \g_@@_delta_x_two_dim
2538   \dim_gzero_new:N \g_@@_delta_y_two_dim
2539 }
2540 \int_zero_new:N \l_@@_initial_i_int
2541 \int_zero_new:N \l_@@_initial_j_int
2542 \int_zero_new:N \l_@@_final_i_int
2543 \int_zero_new:N \l_@@_final_j_int
2544 \bool_set_false:N \l_@@_initial_open_bool
2545 \bool_set_false:N \l_@@_final_open_bool
```

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

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.

```

2546   \bool_if:NT \l_@@_small_bool
2547   {
2548     \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
2549     \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.

```

2550     \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
2551   }

```

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

```

2552   \@@_draw_dotted_lines:

```

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

```

2553   \@@_compute_corners:

```

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

```

2554   \@@_adjust_pos_of_blocks_seq:

```

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

```

2555   \bool_lazy_all:nT
2556   {
2557     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
2558     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
2559     { \seq_if_empty_p:N \l_@@_corners_cells_seq }
2560   }
2561   {
2562     \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn
2563     \cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn
2564   }
2565   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
2566   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
2567   \cs_set_eq:NN \SubMatrix \@@_SubMatrix

```

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

```

2568   \bool_if:NT \c_@@_tikz_loaded_bool
2569   {
2570     \tikzset
2571     {
2572       every~picture / .style =
2573       {
2574         overlay ,
2575         remember~picture ,
2576         name~prefix = \@@_env: -
2577       }
2578     }
2579   }
2580   \cs_set_eq:NN \line \@@_line
2581   \g_@@_internal_code_after_tl
2582   \tl_gclear:N \g_@@_internal_code_after_tl

```


When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it's possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That's why we set `\Code-after` to be *no-op* now.

```
2583 \cs_set_eq:NN \CodeAfter \prg_do_nothing:
```

We clear the list of the names of the potential `\SubMatrix` that will appear in the `\CodeAfter` (unfortunately, that list has to be global).

```
2584 \seq_gclear:N \g_@@_submatrix_names_seq
```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```
2585 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
2586 \scan_stop:
2587 \tl_gclear:N \g_nicematrix_code_after_tl
2588 \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.

```
2589 \tl_if_empty:NF \g_nicematrix_code_before_tl
2590 {
```

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`.

```
2591 \cs_set_protected:Npn \rectanglecolor { }
2592 \cs_set_protected:Npn \columncolor { }
2593 \iow_now:Nn \@mainaux \ExplSyntaxOn
2594 \iow_now:Nx \@mainaux
2595 {
2596 \tl_gset:cn
2597 { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
2598 { \exp_not:V \g_nicematrix_code_before_tl }
2599 }
2600 \iow_now:Nn \@mainaux \ExplSyntaxOff
2601 \bool_set_true:N \l_@@_code_before_bool
2602 }
2603 % \bool_if:NT \l_@@_code_before_bool \@@_write_aux_for_cell_nodes:

2604 \str_gclear:N \g_@@_name_env_str
2605 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁵⁹. 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`.

```
2606 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
2607 }
```

The following command will extract the potential options (between square brackets) at the beginning of the `\CodeAfter` (that is to say, when `\CodeAfter` is used, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```
2608 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
2609 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }
```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format *i-j*. However, the user is allowed to omit *i* or *j* (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block

⁵⁹e.g. `\color[rgb]{0.5,0.5,0}`

(without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It's possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```

2610 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
2611 {
2612     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
2613     { \@@_adjust_pos_of_blocks_seq_i:nnnn #1 }
2614 }

```

The following command must *not* be protected.

```

2615 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnn #1 #2 #3 #4
2616 {
2617     { #1 }
2618     { #2 }
2619     {
2620         \int_compare:nNnTF { #3 } > { 99 }
2621         { \int_use:N \c@iRow }
2622         { #3 }
2623     }
2624     {
2625         \int_compare:nNnTF { #4 } > { 99 }
2626         { \int_use:N \c@jCol }
2627         { #4 }
2628     }
2629 }

```

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).

```

2630 \AtBeginDocument
2631 {
2632     \cs_new_protected:Npx \@@_draw_dotted_lines:
2633     {
2634         \c_@@_pgfortikzpicture_tl
2635         \@@_draw_dotted_lines_i:
2636         \c_@@_endpgfortikzpicture_tl
2637     }
2638 }

```

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

```

2639 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
2640 {
2641     \pgfrememberpicturerepositiononpagetrue
2642     \pgf@relevantforpicturesizefalse
2643     \g_@@_HVdotsfor_lines_tl
2644     \g_@@_Vdots_lines_tl
2645     \g_@@_Ddots_lines_tl
2646     \g_@@_Iddots_lines_tl
2647     \g_@@_Cdots_lines_tl
2648     \g_@@_Ldots_lines_tl
2649 }

2650 \cs_new_protected:Npn \@@_restore_iRow_jCol:
2651 {
2652     \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
2653     \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
2654 }

```

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

```

2655 \pgfdeclareshape { @@_diag_node }

```

```

2656 {
2657   \savedanchor { \five }
2658   {
2659     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
2660     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
2661   }
2662   \anchor { 5 } { \five }
2663   \anchor { center } { \pgfpointorigin }
2664 }

2665 \cs_new_protected:Npn \@@_write_aux_for_cell_nodes:
2666 {
2667   \pgfpicture
2668   \pgfrememberpicturepositiononpagetrue
2669   \pgf@relevantforpicturesizefalse
2670   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2671   {
2672     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2673     {
2674       \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
2675       {
2676         \pgfscope
2677         \pgftransformshift
2678         { \pgfpointanchor { \@@_env: - ##1 - #####1 } { north-west } }
2679         \pgfnode
2680         { rectangle }
2681         { center }
2682         {
2683           \hbox
2684           { \pgfsys@markposition { \@@_env: - ##1 - #####1 - NW } }
2685         }
2686         { }
2687         { }
2688       \endpgfscope
2689       \pgfscope
2690       \pgftransformshift
2691       { \pgfpointanchor { \@@_env: - ##1 - #####1 } { south-east } }
2692       \pgfnode
2693       { rectangle }
2694       { center }
2695       {
2696         \hbox
2697         { \pgfsys@markposition { \@@_env: - ##1 - #####1 - SE } }
2698       }
2699       { }
2700       { }
2701     \endpgfscope
2702   }
2703 }
2704 }
2705 \endpgfpicture
2706 \@@_create_extra_nodes:
2707 }
2708 % \end{macrocode}
2709 %
2710 % \bigskip
2711 % The following command creates the diagonal nodes (in fact, if the matrix is
2712 % not a square matrix, not all the nodes are on the diagonal).
2713 % \begin{macrocode}
2714 \cs_new_protected:Npn \@@_create_diag_nodes:
2715 {
2716   \pgfpicture
2717   \pgfrememberpicturepositiononpagetrue

```

```

2718 \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
2719 {
2720   \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
2721   \dim_set_eq:NN \l_tmpa_dim \pgf@x
2722   \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
2723   \dim_set_eq:NN \l_tmpb_dim \pgf@y
2724   \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
2725   \dim_set_eq:NN \l_tmpc_dim \pgf@x
2726   \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
2727   \dim_set_eq:NN \l_tmpd_dim \pgf@y
2728   \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@â_diag_node`) that we will construct.

```

2729   \dim_set:Nn \l_tmpa_dim { ( \l_tmpc_dim - \l_tmpa_dim ) / 2 }
2730   \dim_set:Nn \l_tmpb_dim { ( \l_tmpd_dim - \l_tmpb_dim ) / 2 }
2731   \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { } }
2732 }

```

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

```

2733 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
2734 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
2735 \dim_set_eq:NN \l_tmpa_dim \pgf@y
2736 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
2737 \pgfcoordinate
2738 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
2739 \pgfnodealias
2740 { \@@_env: - last }
2741 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
2742 \endpgfpicture
2743 }

```

We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

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

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```

2744 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
2745 {

```

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

```
2746 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
2747 \int_set:Nn \l_@@_initial_i_int { #1 }
2748 \int_set:Nn \l_@@_initial_j_int { #2 }
2749 \int_set:Nn \l_@@_final_i_int { #1 }
2750 \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.

```
2751 \bool_set_false:N \l_@@_stop_loop_bool
2752 \bool_do_until:Nn \l_@@_stop_loop_bool
2753 {
2754   \int_add:Nn \l_@@_final_i_int { #3 }
2755   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
2756 \bool_set_false:N \l_@@_final_open_bool
2757 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
2758 {
2759   \int_compare:nNnTF { #3 } = 1
2760   { \bool_set_true:N \l_@@_final_open_bool }
2761   {
2762     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
2763     { \bool_set_true:N \l_@@_final_open_bool }
2764   }
2765 }
2766 {
2767   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
2768   {
2769     \int_compare:nNnT { #4 } = { -1 }
2770     { \bool_set_true:N \l_@@_final_open_bool }
2771   }
2772   {
2773     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
2774     {
2775       \int_compare:nNnT { #4 } = 1
2776       { \bool_set_true:N \l_@@_final_open_bool }
2777     }
2778   }
2779 }
2780 \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.

```
2781 {
```

We do a step backwards.

```
2782 \int_sub:Nn \l_@@_final_i_int { #3 }
2783 \int_sub:Nn \l_@@_final_j_int { #4 }
2784 \bool_set_true:N \l_@@_stop_loop_bool
2785 }
```

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`.

```
2786 {
2787   \cs_if_exist:cTF
2788   {
2789     @@ _ dotted _
2790     \int_use:N \l_@@_final_i_int -
2791     \int_use:N \l_@@_final_j_int
2792   }
2793   {
2794     \int_sub:Nn \l_@@_final_i_int { #3 }
```

```

2795     \int_sub:Nn \l_@@_final_j_int { #4 }
2796     \bool_set_true:N \l_@@_final_open_bool
2797     \bool_set_true:N \l_@@_stop_loop_bool
2798   }
2799   {
2800     \cs_if_exist:cTF
2801     {
2802       pgf @ sh @ ns @ \@@_env:
2803       - \int_use:N \l_@@_final_i_int
2804       - \int_use:N \l_@@_final_j_int
2805     }
2806     { \bool_set_true:N \l_@@_stop_loop_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```

2807     {
2808       \cs_set:cpn
2809       {
2810         @@ _ dotted _
2811         \int_use:N \l_@@_final_i_int -
2812         \int_use:N \l_@@_final_j_int
2813       }
2814       { }
2815     }
2816   }
2817 }
2818 }

```

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

```

2819     \bool_set_false:N \l_@@_stop_loop_bool
2820     \bool_do_until:Nn \l_@@_stop_loop_bool
2821     {
2822       \int_sub:Nn \l_@@_initial_i_int { #3 }
2823       \int_sub:Nn \l_@@_initial_j_int { #4 }
2824       \bool_set_false:N \l_@@_initial_open_bool
2825       \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
2826       {
2827         \int_compare:nNnTF { #3 } = 1
2828         { \bool_set_true:N \l_@@_initial_open_bool }
2829         {
2830           \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
2831           { \bool_set_true:N \l_@@_initial_open_bool }
2832         }
2833       }
2834       {
2835         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
2836         {
2837           \int_compare:nNnT { #4 } = 1
2838           { \bool_set_true:N \l_@@_initial_open_bool }
2839         }
2840         {
2841           \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
2842           {
2843             \int_compare:nNnT { #4 } = { -1 }
2844             { \bool_set_true:N \l_@@_initial_open_bool }
2845           }
2846         }
2847       }

```

```

2848 \bool_if:NTF \l_@@_initial_open_bool
2849 {
2850   \int_add:Nn \l_@@_initial_i_int { #3 }
2851   \int_add:Nn \l_@@_initial_j_int { #4 }
2852   \bool_set_true:N \l_@@_stop_loop_bool
2853 }
2854 {
2855   \cs_if_exist:cTF
2856   {
2857     @@ _ dotted _
2858     \int_use:N \l_@@_initial_i_int -
2859     \int_use:N \l_@@_initial_j_int
2860   }
2861   {
2862     \int_add:Nn \l_@@_initial_i_int { #3 }
2863     \int_add:Nn \l_@@_initial_j_int { #4 }
2864     \bool_set_true:N \l_@@_initial_open_bool
2865     \bool_set_true:N \l_@@_stop_loop_bool
2866   }
2867   {
2868     \cs_if_exist:cTF
2869     {
2870       pgf @ sh @ ns @ \@@_env:
2871       - \int_use:N \l_@@_initial_i_int
2872       - \int_use:N \l_@@_initial_j_int
2873     }
2874     { \bool_set_true:N \l_@@_stop_loop_bool }
2875     {
2876       \cs_set:cpn
2877       {
2878         @@ _ dotted _
2879         \int_use:N \l_@@_initial_i_int -
2880         \int_use:N \l_@@_initial_j_int
2881       }
2882       { }
2883     }
2884   }
2885 }
2886 }

```

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.

```

2887 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
2888 {
2889   { \int_use:N \l_@@_initial_i_int }
2890   { \int_use:N \l_@@_initial_j_int }
2891   { \int_use:N \l_@@_final_i_int }
2892   { \int_use:N \l_@@_final_j_int }
2893 }
2894 }

```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row #1 and column #2. As of now, it’s only the whole array (excepted exterior row and columns).

```

2895 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
2896 {
2897   \int_set:Nn \l_@@_row_min_int 1
2898   \int_set:Nn \l_@@_col_min_int 1
2899   \int_set_eq:NN \l_@@_row_max_int \c@iRow
2900   \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

We do a loop over all the submatrices specified in the `code-before`. We have stored the position of all those submatrices in `\g_@@_submatrix_seq`.

```

2901 \seq_map_inline:Nn \g_@@_submatrix_seq
2902 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
2903 }

```

#1 and #2 are the numbers of row and columns of the cell where the command of dotted line (ex.: \Vdots) has been issued. #3, #4, #5 and #6 are the specification (in *i* and *j*) of the submatrix where are analysing.

```

2904 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
2905 {
2906   \bool_if:nT
2907   {
2908     \int_compare_p:n { #3 <= #1 }
2909     && \int_compare_p:n { #1 <= #5 }
2910     && \int_compare_p:n { #4 <= #2 }
2911     && \int_compare_p:n { #2 <= #6 }
2912   }
2913   {
2914     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
2915     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
2916     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
2917     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
2918   }
2919 }

2920 \cs_new_protected:Npn \@@_set_initial_coords:
2921 {
2922   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2923   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2924 }
2925 \cs_new_protected:Npn \@@_set_final_coords:
2926 {
2927   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2928   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2929 }
2930 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
2931 {
2932   \pgfpointanchor
2933   {
2934     \@@_env:
2935     - \int_use:N \l_@@_initial_i_int
2936     - \int_use:N \l_@@_initial_j_int
2937   }
2938   { #1 }
2939   \@@_set_initial_coords:
2940 }
2941 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
2942 {
2943   \pgfpointanchor
2944   {
2945     \@@_env:
2946     - \int_use:N \l_@@_final_i_int
2947     - \int_use:N \l_@@_final_j_int
2948   }
2949   { #1 }
2950   \@@_set_final_coords:
2951 }

2952 \cs_new_protected:Npn \@@_open_x_initial_dim:
2953 {
2954   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
2955   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2956   {
2957     \cs_if_exist:cT
2958     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }

```



```

2959     {
2960         \pgfpointanchor
2961         { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
2962         { west }
2963         \dim_set:Nn \l_@@_x_initial_dim
2964         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
2965     }
2966 }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

2967     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
2968     {
2969         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2970         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2971         \dim_add:Nn \l_@@_x_initial_dim \col@sep
2972     }
2973 }
2974 \cs_new_protected:Npn \@@_open_x_final_dim:
2975 {
2976     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
2977     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2978     {
2979         \cs_if_exist:cT
2980         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
2981         {
2982             \pgfpointanchor
2983             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
2984             { east }
2985             \dim_set:Nn \l_@@_x_final_dim
2986             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
2987         }
2988     }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

2989     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
2990     {
2991         \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2992         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2993         \dim_sub:Nn \l_@@_x_final_dim \col@sep
2994     }
2995 }

```

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.

```

2996 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
2997 {
2998     \@@_adjust_to_submatrix:nn { #1 } { #2 }
2999     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3000     {
3001         \@@_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.

```

3002     \group_begin:
3003     \int_compare:nNnTF { #1 } = 0
3004     { \color { nicematrix-first-row } }
3005     {

```

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.

```

3006         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3007         { \color { nicematrix-last-row } }
3008     }

```

```

3009         \keys_set:nn { NiceMatrix / xdots } { #3 }
3010         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3011         \@@_actually_draw_Ldots:
3012     \group_end:
3013 }
3014 }

```

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`.

```

3015 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3016 {
3017     \bool_if:NTF \l_@@_initial_open_bool
3018     {
3019         \@@_open_x_initial_dim:
3020         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3021         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3022     }
3023     { \@@_set_initial_coords_from_anchor:n { base-east } }
3024     \bool_if:NTF \l_@@_final_open_bool
3025     {
3026         \@@_open_x_final_dim:
3027         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3028         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3029     }
3030     { \@@_set_final_coords_from_anchor:n { base-west } }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of text. Of course, maybe we should not do that when the option `line-style` is used (?).

```

3031     \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3032     \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3033     \@@_draw_line:
3034 }

```

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.

```

3035 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3036 {
3037     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3038     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3039     {
3040         \@@_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.

```

3041     \group_begin:
3042     \int_compare:nNnTF { #1 } = 0
3043     { \color { nicematrix-first-row } }
3044     {

```

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.

```

3045         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3046         { \color { nicematrix-last-row } }
3047     }
3048     \keys_set:nn { NiceMatrix / xdots } { #3 }
3049     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3050     \@@_actually_draw_Cdots:
3051 \group_end:
3052 }
3053 }

```

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`.

```

3054 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3055 {
3056     \bool_if:NTF \l_@@_initial_open_bool
3057     { \@@_open_x_initial_dim: }
3058     { \@@_set_initial_coords_from_anchor:n { mid-east } }
3059     \bool_if:NTF \l_@@_final_open_bool
3060     { \@@_open_x_final_dim: }
3061     { \@@_set_final_coords_from_anchor:n { mid-west } }
3062     \bool_lazy_and:nnTF
3063     \l_@@_initial_open_bool
3064     \l_@@_final_open_bool
3065     {
3066         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3067         \dim_set_eq:NN \l_tmpa_dim \pgf@y
3068         \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
3069         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3070         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3071     }
3072     {
3073         \bool_if:NT \l_@@_initial_open_bool
3074         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3075         \bool_if:NT \l_@@_final_open_bool
3076         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3077     }
3078     \@@_draw_line:
3079 }
3080 \cs_new_protected:Npn \@@_open_y_initial_dim:
3081 {
3082     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3083     \dim_set:Nn \l_@@_y_initial_dim
3084     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3085     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3086     {
3087         \cs_if_exist:cT
3088         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3089         {
3090             \pgfpointanchor
3091             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }

```

```

3092         { north }
3093         \dim_set:Nn \l_@@_y_initial_dim
3094         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3095     }
3096 }
3097 }
3098 \cs_new_protected:Npn \@@_open_y_final_dim:
3099 {
3100     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3101     \dim_set:Nn \l_@@_y_final_dim
3102     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3103     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3104     {
3105         \cs_if_exist:cT
3106         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3107         {
3108             \pgfpointanchor
3109             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3110             { south }
3111             \dim_set:Nn \l_@@_y_final_dim
3112             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3113         }
3114     }
3115 }

```

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.

```

3116 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3117 {
3118     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3119     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3120     {
3121         \@@_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.

```

3122     \group_begin:
3123     \int_compare:nNnTF { #2 } = 0
3124     { \color { nicematrix-first-col } }
3125     {
3126         \int_compare:nNnT { #2 } = \l_@@_last_col_int
3127         { \color { nicematrix-last-col } }
3128     }
3129     \keys_set:nn { NiceMatrix / xdots } { #3 }
3130     \tl_if_empty:VF \l_@@_xdots_color_tl
3131     { \color { \l_@@_xdots_color_tl } }
3132     \@@_actually_draw_Vdots:
3133     \group_end:
3134 }
3135 }

```

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`.

```
3136 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3137 {
```

The boolean `\l_tmpa_bool` indicates whether the column is of type `l` or may be considered as if.

```
3138 \bool_set_false:N \l_tmpa_bool
```

First the case when the line is closed on both ends.

```
3139 \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3140 {
3141   \@@_set_initial_coords_from_anchor:n { south-west }
3142   \@@_set_final_coords_from_anchor:n { north-west }
3143   \bool_set:Nn \l_tmpa_bool
3144   { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3145 }
```

Now, we try to determine whether the column is of type `c` or may be considered as if.

```
3146 \bool_if:NTF \l_@@_initial_open_bool
3147   \@@_open_y_initial_dim:
3148   { \@@_set_initial_coords_from_anchor:n { south } }
3149 \bool_if:NTF \l_@@_final_open_bool
3150   \@@_open_y_final_dim:
3151   { \@@_set_final_coords_from_anchor:n { north } }
3152 \bool_if:NTF \l_@@_initial_open_bool
3153 {
3154   \bool_if:NTF \l_@@_final_open_bool
3155   {
3156     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3157     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3158     \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3159     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3160     \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.

```
3161 \int_compare:nNnT \l_@@_last_col_int > { -2 }
3162 {
3163   \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3164   {
3165     \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3166     \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3167     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3168     \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3169   }
3170 }
3171 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3172 }
3173 {
3174   \bool_if:NTF \l_@@_final_open_bool
3175   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3176 }
```

Now the case where both extremities are closed. The first conditional tests whether the column is of type `c` or may be considered as if.

```
3178 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3179 {
3180   \dim_set:Nn \l_@@_x_initial_dim
3181   {
3182     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3183     \l_@@_x_initial_dim \l_@@_x_final_dim
3184   }
3185   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3186 }
```

```

3187     }
3188   }
3189   \@@_draw_line:
3190 }

```

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.

```

3191 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3192 {
3193   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3194   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3195   {
3196     \@@_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.

```

3197     \group_begin:
3198     \keys_set:nn { NiceMatrix / xdots } { #3 }
3199     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3200     \@@_actually_draw_Ddots:
3201   \group_end:
3202 }
3203 }

```

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`.

```

3204 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3205 {
3206   \bool_if:NTF \l_@@_initial_open_bool
3207   {
3208     \@@_open_y_initial_dim:
3209     % \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3210     % \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3211     \@@_open_x_initial_dim:
3212   }
3213   { \@@_set_initial_coords_from_anchor:n { south-east } }
3214   \bool_if:NTF \l_@@_final_open_bool
3215   {
3216     % \@@_open_y_final_dim:
3217     % \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3218     \@@_open_x_final_dim:
3219     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3220   }
3221   { \@@_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.

```

3222   \bool_if:NT \l_@@_parallelize_diags_bool
3223   {
3224     \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).

```
3225 \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.

```
3226 {
3227   \dim_gset:Nn \g_@@_delta_x_one_dim
3228   { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3229   \dim_gset:Nn \g_@@_delta_y_one_dim
3230   { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3231 }
```

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`.

```
3232 {
3233   \dim_set:Nn \l_@@_y_final_dim
3234   {
3235     \l_@@_y_initial_dim +
3236     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3237     \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3238   }
3239 }
3240 }
3241 \@@_draw_line:
3242 }
```

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.

```
3243 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3244 {
3245   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3246   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3247   {
3248     \@@_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.

```
3249   \group_begin:
3250   \keys_set:nn { NiceMatrix / xdots } { #3 }
3251   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3252   \@@_actually_draw_Iddots:
3253   \group_end:
3254 }
3255 }
```

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`.

```

3256 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3257 {
3258   \bool_if:NTF \l_@@_initial_open_bool
3259   {
3260     \@@_open_y_initial_dim:
3261     \@@_open_x_initial_dim:
3262   }
3263   { \@@_set_initial_coords_from_anchor:n { south-west } }
3264   \bool_if:NTF \l_@@_final_open_bool
3265   {
3266     \@@_open_y_final_dim:
3267     \@@_open_x_final_dim:
3268   }
3269   { \@@_set_final_coords_from_anchor:n { north-east } }
3270   \bool_if:NT \l_@@_parallelize_diags_bool
3271   {
3272     \int_gincr:N \g_@@_iddots_int
3273     \int_compare:nNnTF \g_@@_iddots_int = 1
3274     {
3275       \dim_gset:Nn \g_@@_delta_x_two_dim
3276       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3277       \dim_gset:Nn \g_@@_delta_y_two_dim
3278       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3279     }
3280     {
3281       \dim_set:Nn \l_@@_y_final_dim
3282       {
3283         \l_@@_y_initial_dim +
3284         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3285         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3286       }
3287     }
3288   }
3289   \@@_draw_line:
3290 }

```

The actual instructions for drawing the dotted line with Tikz

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

3291 \cs_new_protected:Npn \@@_draw_line:
3292 {
3293   \pgfrememberpicturepositiononpagetrue
3294   \pgf@relevantforpicturesizefalse
3295   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
3296   \@@_draw_standard_dotted_line:
3297   \@@_draw_non_standard_dotted_line:
3298 }

```


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.

```

3299 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
3300 {
3301   \begin { scope }
3302   \exp_args:No \@@_draw_non_standard_dotted_line:n
3303     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3304 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_non_standard_dotted_line:n` is, in fact, the list of options.

```

3305 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
3306 {
3307   \@@_draw_non_standard_dotted_line:nVV
3308     { #1 }
3309   \l_@@_xdots_up_tl
3310   \l_@@_xdots_down_tl
3311 }

3312 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:nnn #1 #2 #3
3313 {
3314   \draw
3315     [
3316       #1 ,
3317       shorten~> = \l_@@_xdots_shorten_dim ,
3318       shorten~< = \l_@@_xdots_shorten_dim ,
3319     ]
3320     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )

```

Be careful: We can't put `\c_math_toggle_token` instead of `$` in the following lines because we are in the contents of Tikz nodes (and they will be *rescanned* if the Tikz library `babel` is loaded).

```

3321   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3322   node [ sloped , below ] { $ \scriptstyle #3 $ }
3323   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3324   \end { scope }
3325 }
3326 \cs_generate_variant:Nn \@@_draw_non_standard_dotted_line:nnn { n V V }

```

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

```

3327 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3328 {
3329   \bool_lazy_and:nnF
3330     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3331     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3332   {
3333     \pgfscope
3334     \pgftransformshift
3335       {
3336         \pgfpointlineattime { 0.5 }
3337         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3338         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3339       }
3340     \pgftransformrotate
3341       {
3342         \fp_eval:n
3343         {
3344           atand
3345             (
3346               \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3347               \l_@@_x_final_dim - \l_@@_x_initial_dim
3348             )

```

```

3349         }
3350     }
3351     \pgfnode
3352     { rectangle }
3353     { south }
3354     {
3355         \c_math_toggle_token
3356         \scriptstyle \l_@@_xdots_up_tl
3357         \c_math_toggle_token
3358     }
3359     { }
3360     { \pgfusepath { } }
3361     \pgfnode
3362     { rectangle }
3363     { north }
3364     {
3365         \c_math_toggle_token
3366         \scriptstyle \l_@@_xdots_down_tl
3367         \c_math_toggle_token
3368     }
3369     { }
3370     { \pgfusepath { } }
3371     \endpgfscope
3372 }
3373 \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.

```

3374     \dim_zero_new:N \l_@@_l_dim
3375     \dim_set:Nn \l_@@_l_dim
3376     {
3377         \fp_to_dim:n
3378         {
3379             sqrt
3380             (
3381                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3382                 +
3383                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3384             )
3385         }
3386     }

```

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.

```

3387     \bool_lazy_or:nnF
3388     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3389     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3390     \@@_draw_standard_dotted_line_i:
3391     \group_end:
3392 }
3393 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3394 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3395 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

3396     \bool_if:NTF \l_@@_initial_open_bool
3397     {
3398         \bool_if:NTF \l_@@_final_open_bool
3399         {
3400             \int_set:Nn \l_tmpa_int
3401             { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }

```

```

3402     }
3403     {
3404         \int_set:Nn \l_tmpa_int
3405         {
3406             \dim_ratio:nn
3407             { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3408             \l_@@_inter_dots_dim
3409         }
3410     }
3411 }
3412 {
3413     \bool_if:NTF \l_@@_final_open_bool
3414     {
3415         \int_set:Nn \l_tmpa_int
3416         {
3417             \dim_ratio:nn
3418             { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3419             \l_@@_inter_dots_dim
3420         }
3421     }
3422     {
3423         \int_set:Nn \l_tmpa_int
3424         {
3425             \dim_ratio:nn
3426             { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3427             \l_@@_inter_dots_dim
3428         }
3429     }
3430 }

```

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

```

3431     \dim_set:Nn \l_tmpa_dim
3432     {
3433         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3434         \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3435     }
3436     \dim_set:Nn \l_tmpb_dim
3437     {
3438         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3439         \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3440     }

```

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`.

```

3441     \int_set:Nn \l_tmpb_int
3442     {
3443         \bool_if:NTF \l_@@_initial_open_bool
3444         { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3445         { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3446     }

```

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.

```

3447     \dim_gadd:Nn \l_@@_x_initial_dim
3448     {
3449         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3450         \dim_ratio:nn
3451         { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3452         { 2 \l_@@_l_dim }
3453         * \l_tmpb_int
3454     }

```

```

3455 \dim_gadd:Nn \l_@@_y_initial_dim
3456 {
3457   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3458   \dim_ratio:nn
3459   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3460   { 2 \l_@@_l_dim }
3461   * \l_tmpb_int
3462 }
3463 \pgf@relevantforpicturesizefalse
3464 \int_step_inline:nnn 0 \l_tmpa_int
3465 {
3466   \pgfpathcircle
3467   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3468   { \l_@@_radius_dim }
3469   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3470   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3471 }
3472 \pgfusepathqfill
3473 }

```

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.

```

3474 \AtBeginDocument
3475 {
3476   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3477   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3478   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3479   {
3480     \int_compare:nNnTF \c@jCol = 0
3481     { \@@_error:nn { in~first~col } \Ldots }
3482     {
3483       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3484       { \@@_error:nn { in~last~col } \Ldots }
3485       {
3486         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
3487         { #1 , down = #2 , up = #3 }
3488       }
3489     }
3490     \bool_if:NF \l_@@_nullify_dots_bool
3491     { \phantom { \ensuremath { \@@_old_ldots } } }
3492     \bool_gset_true:N \g_@@_empty_cell_bool
3493   }

3494   \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3495   {
3496     \int_compare:nNnTF \c@jCol = 0
3497     { \@@_error:nn { in~first~col } \Cdots }

```

```

3498     {
3499         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3500         { \@@_error:nn { in~last~col } \Cdots }
3501         {
3502             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
3503             { #1 , down = #2 , up = #3 }
3504         }
3505     }
3506     \bool_if:NF \l_@@_nullify_dots_bool
3507     { \phantom { \ensuremath { \@@_old_cdots } } }
3508     \bool_gset_true:N \g_@@_empty_cell_bool
3509 }

\exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
{
3512     \int_compare:nNnTF \c@iRow = 0
3513     { \@@_error:nn { in~first~row } \Vdots }
3514     {
3515         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
3516         { \@@_error:nn { in~last~row } \Vdots }
3517         {
3518             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
3519             { #1 , down = #2 , up = #3 }
3520         }
3521     }
3522     \bool_if:NF \l_@@_nullify_dots_bool
3523     { \phantom { \ensuremath { \@@_old_vdots } } }
3524     \bool_gset_true:N \g_@@_empty_cell_bool
3525 }

\exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
{
3528     \int_case:nnF \c@iRow
3529     {
3530         0 { \@@_error:nn { in~first~row } \Ddots }
3531         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
3532     }
3533     {
3534         \int_case:nnF \c@jCol
3535         {
3536             0 { \@@_error:nn { in~first~col } \Ddots }
3537             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
3538         }
3539         {
3540             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3541             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
3542             { #1 , down = #2 , up = #3 }
3543         }
3544     }
3545 }
3546 \bool_if:NF \l_@@_nullify_dots_bool
3547 { \phantom { \ensuremath { \@@_old_ddots } } }
3548 \bool_gset_true:N \g_@@_empty_cell_bool
3549 }

\exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
{
3552     \int_case:nnF \c@iRow
3553     {
3554         0 { \@@_error:nn { in~first~row } \Iddots }
3555         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }

```

```

3556     }
3557     {
3558         \int_case:nnF \c@jCol
3559         {
3560             0 { \@@_error:nn { in~first~col } \Iddots }
3561             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
3562         }
3563         {
3564             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3565             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
3566             { #1 , down = #2 , up = #3 }
3567         }
3568     }
3569     \bool_if:NF \l_@@_nullify_dots_bool
3570     { \phantom { \ensuremath { \@@_old_iddots } } }
3571     \bool_gset_true:N \g_@@_empty_cell_bool
3572 }
3573 }

```

End of the \AtBeginDocument.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

3574 \keys_define:nn { NiceMatrix / Ddots }
3575 {
3576     draw-first .bool_set:N = \l_@@_draw_first_bool ,
3577     draw-first .default:n = true ,
3578     draw-first .value_forbidden:n = true
3579 }

```

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

```

3580 \cs_new_protected:Npn \@@_Hspace:
3581 {
3582     \bool_gset_true:N \g_@@_empty_cell_bool
3583     \hspace
3584 }

```

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

```

3585 \cs_set_eq:NN \@@_old_multicolumn \multicolumn
3586 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3587 {

```

We have to act in an expandable way since it will begin by a \multicolumn.

```

3588     \exp_args:NNe
3589     \@@_old_multicolumn
3590     { #1 }
3591     {
3592         \exp_args:Ne \str_case:nn { \str_foldcase:n { #2 } }
3593         {
3594             l { > \@@_Cell: l < \@@_end_Cell: }
3595             r { > \@@_Cell: r < \@@_end_Cell: }
3596             c { > \@@_Cell: c < \@@_end_Cell: }
3597             { l | } { > \@@_Cell: l < \@@_end_Cell: | }
3598             { r | } { > \@@_Cell: r < \@@_end_Cell: | }
3599             { c | } { > \@@_Cell: c < \@@_end_Cell: | }
3600             { | l } { | > \@@_Cell: l < \@@_end_Cell: }
3601             { | r } { | > \@@_Cell: r < \@@_end_Cell: }
3602             { | c } { | > \@@_Cell: c < \@@_end_Cell: }
3603             { | l | } { | > \@@_Cell: l < \@@_end_Cell: | }
3604             { | r | } { | > \@@_Cell: r < \@@_end_Cell: | }
3605             { | c | } { | > \@@_Cell: c < \@@_end_Cell: | }
3606         }
3607     }
3608     { #3 }

```

The `\peek_remove_spaces:n` is mandatory.

```

3609   \peek_remove_spaces:n
3610   {
3611     \int_compare:nNnT #1 > 1
3612     {
3613       \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
3614       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3615       \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
3616       \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
3617       {
3618         { \int_use:N \c@iRow }
3619         { \int_use:N \c@jCol }
3620         { \int_use:N \c@iRow }
3621         { \int_eval:n { \c@jCol + #1 - 1 } }
3622       }
3623     }
3624     \int_gadd:Nn \c@jCol { #1 - 1 }
3625     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3626     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3627   }
3628 }

```

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`.

```

3629 \cs_new:Npn \@@_Hdotsfor:
3630 {
3631   \bool_lazy_and:nnTF
3632   { \int_compare_p:nNn \c@jCol = 0 }
3633   { \int_compare_p:nNn \l_@@_first_col_int = 0 }
3634   {
3635     \bool_if:NTF \g_@@_after_col_zero_bool
3636     {
3637       \multicolumn { 1 } { c } { }
3638       \@@_Hdotsfor_i
3639     }
3640     { \@@_fatal:n { Hdotsfor~in~col~0 } }
3641   }
3642   {
3643     \multicolumn { 1 } { c } { }
3644     \@@_Hdotsfor_i
3645   }
3646 }

```

The command `\@@_Hdotsfor_i` is defined with `\NewDocumentCommand` because it has an optional argument. Note that such a command defined by `\NewDocumentCommand` is protected and that's why we have put the `\multicolumn` before (in the definition of `\@@_Hdotsfor:`).

```

3647 \AtBeginDocument
3648 {
3649   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3650   \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.

```

3651   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
3652   {
3653     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
3654     {
3655       \@@_Hdotsfor:nnnn
3656       { \int_use:N \c@iRow }
3657       { \int_use:N \c@jCol }
3658       { #2 }
3659     }

```

```

3660         #1 , #3 ,
3661         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3662     }
3663 }
3664 \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
3665 }
3666 }

```

Enf of \AtBeginDocument.

```

3667 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
3668 {
3669     \bool_set_false:N \l_@@_initial_open_bool
3670     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

3671     \int_set:Nn \l_@@_initial_i_int { #1 }
3672     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

3673     \int_compare:nNnTF { #2 } = 1
3674     {
3675         \int_set:Nn \l_@@_initial_j_int 1
3676         \bool_set_true:N \l_@@_initial_open_bool
3677     }
3678     {
3679         \cs_if_exist:cTF
3680         {
3681             pgf @ sh @ ns @ \@@_env:
3682             - \int_use:N \l_@@_initial_i_int
3683             - \int_eval:n { #2 - 1 }
3684         }
3685         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
3686         {
3687             \int_set:Nn \l_@@_initial_j_int { #2 }
3688             \bool_set_true:N \l_@@_initial_open_bool
3689         }
3690     }
3691     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
3692     {
3693         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3694         \bool_set_true:N \l_@@_final_open_bool
3695     }
3696     {
3697         \cs_if_exist:cTF
3698         {
3699             pgf @ sh @ ns @ \@@_env:
3700             - \int_use:N \l_@@_final_i_int
3701             - \int_eval:n { #2 + #3 }
3702         }
3703         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
3704         {
3705             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3706             \bool_set_true:N \l_@@_final_open_bool
3707         }
3708     }
3709     \group_begin:
3710     \int_compare:nNnTF { #1 } = 0
3711     { \color { nicematrix-first-row } }
3712     {
3713         \int_compare:nNnT { #1 } = \g_@@_row_total_int
3714         { \color { nicematrix-last-row } }
3715     }
3716     \keys_set:nn { NiceMatrix / xdots } { #4 }

```



```

3717 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3718 \@@_actually_draw_ldots:
3719 \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).

```

3720 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
3721 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
3722 }

3723 \AtBeginDocument
3724 {
3725   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3726   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3727   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
3728   {
3729     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
3730     {
3731       \@@_Vdotsfor:nnnn
3732       { \int_use:N \c@iRow }
3733       { \int_use:N \c@jCol }
3734       { #2 }
3735       {
3736         #1 , #3 ,
3737         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3738       }
3739     }
3740   }
3741 }

```

Enf of `\AtBeginDocument`.

```

3742 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
3743 {
3744   \bool_set_false:N \l_@@_initial_open_bool
3745   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

3746 \int_set:Nn \l_@@_initial_j_int { #2 }
3747 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

3748 \int_compare:nNnTF #1 = 1
3749 {
3750   \int_set:Nn \l_@@_initial_i_int 1
3751   \bool_set_true:N \l_@@_initial_open_bool
3752 }
3753 {
3754   \cs_if_exist:cTF
3755   {
3756     pgf @ sh @ ns @ \@@_env:
3757     - \int_eval:n { #1 - 1 }
3758     - \int_use:N \l_@@_initial_j_int
3759   }
3760   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
3761   {
3762     \int_set:Nn \l_@@_initial_i_int { #1 }
3763     \bool_set_true:N \l_@@_initial_open_bool
3764   }
3765 }
3766 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
3767 {
3768   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }

```

```

3769     \bool_set_true:N \l_@@_final_open_bool
3770   }
3771   {
3772     \cs_if_exist:cTF
3773     {
3774       pgf @ sh @ ns @ \@@_env:
3775       - \int_eval:n { #1 + #3 }
3776       - \int_use:N \l_@@_final_j_int
3777     }
3778     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
3779     {
3780       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3781       \bool_set_true:N \l_@@_final_open_bool
3782     }
3783   }
3784   \group_begin:
3785   \int_compare:nNnTF { #2 } = 0
3786   { \color { nicematrix-first-col } }
3787   {
3788     \int_compare:nNnT { #2 } = \g_@@_col_total_int
3789     { \color { nicematrix-last-col } }
3790   }
3791   \keys_set:nn { NiceMatrix / xdots } { #4 }
3792   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3793   \@@_actually_draw_Vdots:
3794   \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).

```

3795   \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
3796   { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
3797 }

```

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

```

3798 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

```

The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells.

First, we write a command with an argument of the format i - j and applies the command `\int_eval:n` to i and j ; this must *not* be protected (and is, of course fully expandable).⁶⁰

```

3799 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
3800 { \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).

```

3801 \AtBeginDocument
3802 {
3803   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }

```

⁶⁰Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

3804 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3805 \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
3806 {
3807   \group_begin:
3808   \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
3809   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3810   \use:e
3811   {
3812     \@@_line_i:nn
3813     { \@@_double_int_eval:n #2 \q_stop }
3814     { \@@_double_int_eval:n #3 \q_stop }
3815   }
3816   \group_end:
3817 }
3818 }

3819 \cs_new_protected:Npn \@@_line_i:nn #1 #2
3820 {
3821   \bool_set_false:N \l_@@_initial_open_bool
3822   \bool_set_false:N \l_@@_final_open_bool
3823   \bool_if:nTF
3824   {
3825     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
3826     ||
3827     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
3828   }
3829   {
3830     \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
3831   }
3832   { \@@_draw_line_ii:nn { #1 } { #2 } }
3833 }

3834 \AtBeginDocument
3835 {
3836   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
3837   {

```

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:`.

```

3838   \c_@@_pgfortikzpicture_tl
3839   \@@_draw_line_iii:nn { #1 } { #2 }
3840   \c_@@_endpgfortikzpicture_tl
3841 }
3842 }

```

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

```

3843 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
3844 {
3845   \pgfrememberpicturepositiononpagetrue
3846   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
3847   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3848   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3849   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
3850   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3851   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3852   \@@_draw_line:
3853 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That's why we try to draw rectangles of the same color in the same instruction `\pgfusepath { fill }` (and they will be in the same instruction `fill`—coded `f`— in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor` (which are linked to `\rowcolor`, `\columncolor` and `\rectanglecolor` before the execution of the `code-before`) don't directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence `\g_@@_colors_seq` will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: `[gray]{0.5}`).
- For the color whose index in `\g_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\g_@@_color_i_tl`. In that token list, the instructions will be written using `\@@_rowcolor:n`, `\@@_columncolor:n` and `\@@_rectanglecolor:nn` (corresponding of `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor`).

`bigskip #1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_color_seq` doesn't only add a color to `\g_@@_colors_seq`: it also updates the corresponding token list `\g_@@_color_i_tl`. We add in a global way because the final user may use the instructions such as `\cellcolor` in a loop of `pgffor` in the `\CodeBefore` (and we recall that a loop of `pgffor` is encapsulated in a group).

```
3854 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
3855 {
```

First, we look for the number of the color and, if it's found, we store it in `\l_tmpa_int`. If the color is not present in `\l_@@_colors_seq`, `\l_tmpa_int` will remain equal to 0.

```
3856   \int_zero:N \l_tmpa_int
3857   \seq_map_indexed_inline:Nn \g_@@_colors_seq
3858     { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##2 } } }
3859   \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

First, the case where the color is a *new* color (not in the sequence).

```
3860     {
3861       \seq_gput_right:Nn \g_@@_colors_seq { #1 }
3862       \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
3863     }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
3864     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
3865   }
3866 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```
3867 \cs_new_protected:Npn \@@_actually_color:
3868 {
3869   \pgfpicture
3870   \pgf@relevantforpicturesizefalse
3871   \seq_map_indexed_inline:Nn \g_@@_colors_seq
3872     {
3873       \color ##2
3874       \use:c { g_@@_color _ ##1 _ tl }
3875       \tl_gclear:c { g_@@_color _ ##1 _ tl }
3876       \pgfusepath { fill }
3877     }
3878   \endpgfpicture
3879 }
```

```

3880 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
3881 {
3882   \tl_set:Nn \l_tmpa_tl { #1 }
3883   \tl_set:Nn \l_tmpb_tl { #2 }
3884 }

```

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

```

3885 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
3886 {
3887   \tl_if_blank:nF { #2 }
3888   {
3889     \@@_add_to_colors_seq:xn
3890     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3891     { \@@_rowcolor:n { #3 } }
3892   }
3893 }
3894 \cs_new_protected:Npn \@@_rowcolor:n #1
3895 {
3896   \tl_set:Nn \l_@@_rows_tl { #1 }
3897   \tl_set:Nn \l_@@_cols_tl { - }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

3898   \@@_cartesian_path:
3899 }

```

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

```

3900 \NewDocumentCommand \@@_columncolor { 0 { } m m }
3901 {
3902   \tl_if_blank:nF { #2 }
3903   {
3904     \@@_add_to_colors_seq:xn
3905     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3906     { \@@_columncolor:n { #3 } }
3907   }
3908 }
3909 \cs_new_protected:Npn \@@_columncolor:n #1
3910 {
3911   \tl_set:Nn \l_@@_rows_tl { - }
3912   \tl_set:Nn \l_@@_cols_tl { #1 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

3913   \@@_cartesian_path:
3914 }

```

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

```

3915 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
3916 {
3917   \tl_if_blank:nF { #2 }
3918   {
3919     \@@_add_to_colors_seq:xn
3920     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3921     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
3922   }
3923 }

```

The last argument is the radius of the corners of the rectangle.

```

3924 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
3925 {
3926   \tl_if_blank:nF { #2 }
3927   {
3928     \@@_add_to_colors_seq:xn
3929     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3930     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
3931   }
3932 }

```

The last argument is the radius of the corners of the rectangle.

```

3933 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
3934 {
3935   \@@_cut_on_hyphen:w #1 \q_stop
3936   \tl_clear_new:N \l_tmpc_tl
3937   \tl_clear_new:N \l_tmpd_tl
3938   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
3939   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
3940   \@@_cut_on_hyphen:w #2 \q_stop
3941   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
3942   \tl_set:Nx \l_@@_cols_tl { \l_tmpd_tl - \l_tmpb_tl }

```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

3943   \@@_cartesian_path:n { #3 }
3944 }

```

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

```

3945 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
3946 {
3947   \clist_map_inline:nn { #3 }
3948   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
3949 }

```

```

3950 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
3951 {
3952   \int_step_inline:nn { \int_use:N \c@iRow }
3953   {
3954     \int_step_inline:nn { \int_use:N \c@jCol }
3955     {
3956       \int_if_even:nTF { ####1 + ##1 }
3957       { \@@_cellcolor [ #1 ] { #2 } }
3958       { \@@_cellcolor [ #1 ] { #3 } }
3959       { ##1 - ####1 }
3960     }
3961   }
3962 }

```

```

3963 \keys_define:nn { NiceMatrix / arraycolor }
3964 { except-corners .code:n = \@@_error:n { key except-corners } }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns). The third argument is a optional argument which a list of pairs key-value. As for now, there is only one key: `except-corners`. When that key is used, the cells in the corners are not colored.

```

3965 \NewDocumentCommand \@@_arraycolor { 0 { } m 0 { } }
3966 {
3967   \keys_set:nn { NiceMatrix / arraycolor } { #3 }
3968   \@@_rectanglecolor [ #1 ] { #2 }

```

```

3969     { 1 - 1 }
3970     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3971 }

3972 \keys_define:nn { NiceMatrix / rowcolors }
3973 {
3974     respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
3975     respect-blocks .default:n = true ,
3976     cols .tl_set:N = \l_@@_cols_tl ,
3977     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
3978     restart .default:n = true ,
3979     unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
3980 }

```

The command `\rowcolors` (accessible in the code-before) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`. Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

#1 (optional) is the color space ; #2 is a list of intervals of rows ; #3 is the first color ; #4 is the second color ; #5 is for the optional list of pairs key-value.

```

3981 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
3982 {

```

The group is for the options.

```

3983     \group_begin:
3984     \tl_clear_new:N \l_@@_cols_tl
3985     \tl_set:Nn \l_@@_cols_tl { - }
3986     \keys_set:nn { NiceMatrix / rowcolors } { #5 }

```

The boolean `\l_tmpa_bool` will indicate whereas we are in a row of the first color or of the second color.

```

3987     \bool_set_true:N \l_tmpa_bool
3988     \bool_lazy_and:nnT
3989         \l_@@_respect_blocks_bool
3990         {
3991             \cs_if_exist_p:c
3992                 { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } }
3993     }

```

We don't want to take into account a block which is completely in the “first column” of (number 0) or in the “last column” and that's why we filter the sequence of the blocks (in a the sequence `\l_tmpa_seq`).

```

3994     \seq_set_eq:Nc \l_tmpb_seq
3995         { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
3996     \seq_set_filter:Nn \l_tmpa_seq \l_tmpb_seq
3997         { \@@_not_in_exterior_p:n n n n #1 }
3998     }
3999     \pgfpicture
4000     \pgf@relevantforpicturesizefalse
4001     \clist_map_inline:nn { #2 }
4002     {
4003         \tl_set:Nn \l_tmpa_tl { #1 }
4004         \tl_if_in:NnTF \l_tmpa_tl { - }
4005             { \@@_cut_on_hyphen:w #1 \q_stop }
4006             { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

The counter `\l_tmpa_int` will be the index of the loop.

```

4007     \int_set:Nn \l_tmpa_int \l_tmpa_tl
4008     \bool_if:NTF \l_@@_rowcolors_restart_bool
4009         { \bool_set_true:N \l_tmpa_bool }
4010         { \bool_set:Nn \l_tmpa_bool { \int_if_odd_p:n { \l_tmpa_tl } } }
4011     \int_zero_new:N \l_tmpc_int
4012     \int_set:Nn \l_tmpc_int \l_tmpb_tl

```

```

4013     \int_do_until:nNnn \l_tmpa_int > \l_tmpc_int
4014     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4015     \int_set_eq:NN \l_tmpb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

4016     \bool_lazy_and:nnT
4017     \l_@@_respect_blocks_bool
4018     {
4019         \cs_if_exist_p:c
4020         { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
4021     }
4022     {
4023         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
4024         { \@@_intersect_our_row_p:nnnn ###1 }
4025         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4026     }
4027     \tl_set:Nx \l_@@_rows_tl
4028     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
4029     \bool_if:NTF \l_tmpa_bool
4030     {
4031         \tl_if_blank:nF { #3 }
4032         {
4033             \tl_if_empty:nTF { #1 }
4034             \color
4035             { \color [ #1 ] }
4036             { #3 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4037         \@@_cartesian_path:
4038         \pgfusepath { fill }
4039     }
4040     \bool_set_false:N \l_tmpa_bool
4041 }
4042 {
4043     \tl_if_blank:nF { #4 }
4044     {
4045         \tl_if_empty:nTF { #1 }
4046         \color
4047         { \color [ #1 ] }
4048         { #4 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

4049         \@@_cartesian_path:
4050         \pgfusepath { fill }
4051     }
4052     \bool_set_true:N \l_tmpa_bool
4053 }
4054     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4055 }
4056 }
4057 \endpgfpicture
4058 \group_end:
4059 }

```

```

4060 \cs_new_protected:Npn \@@_rowcolors_i:nnnn #1 #2 #3 #4
4061 {
4062     \int_compare:nNnT { #3 } > \l_tmpb_int
4063     { \int_set:Nn \l_tmpb_int { #3 } }
4064 }

```



```

4065 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnn p
4066 {
4067   \bool_lazy_or:nnTF
4068   { \int_compare_p:nNn { #4 } = \c_zero_int }
4069   { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } } }
4070   \prg_return_false:
4071   \prg_return_true:
4072 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

4073 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnn p
4074 {
4075   \bool_if:nTF
4076   {
4077     \int_compare_p:n { #1 <= \l_tmpa_int }
4078     &&
4079     \int_compare_p:n { \l_tmpa_int <= #3 }
4080   }
4081   \prg_return_true:
4082   \prg_return_false:
4083 }

```

The following command uses two implicit arguments: \l_@@_rows_tl and \l_@@_cols_tl which are specifications for a set of rows and a set of columns. It creates a path but does *not* fill it. It must be filled by another command after. The argument is the radius of the corners. We define below a command \@@_cartesian_path: which corresponds to a value 0 pt for the radius of the corners. This command is in particular used in \@@_rectanglecolor:nnn (used in \@@_rectanglecolor, itself used in \@@_cellcolor).

```

4084 \cs_new_protected:Npn \@@_cartesian_path:n #1
4085 {
4086   \bool_lazy_and:nnT
4087   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4088   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4089   {
4090     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4091     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4092   }

```

We begin the loop over the columns.

```

4093 \clist_map_inline:Nn \l_@@_cols_tl
4094 {
4095   \tl_set:Nn \l_tmpa_tl { ##1 }
4096   \tl_if_in:NnTF \l_tmpa_tl { - }
4097   { \@@_cut_on_hyphen:w ##1 \q_stop }
4098   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4099   \bool_lazy_or:nnT
4100   { \tl_if_blank_p:V \l_tmpa_tl }
4101   { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4102   { \tl_set:Nn \l_tmpa_tl { 1 } }
4103   \bool_lazy_or:nnT
4104   { \tl_if_blank_p:V \l_tmpb_tl }
4105   { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4106   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4107   \int_compare:nNnT \l_tmpb_tl > \c@jCol
4108   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

\l_tmpc_tl will contain the number of column.

```

4109 \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl

```

If we decide to provide the commands \cellcolor, \rectanglecolor, \rowcolor, \columncolor, \rowcolors and \chessboardcolors in the code-before of a \SubMatrix, we will have to modify the following line, by adding a kind of offset. We will have also some other lines to modify.

```

4110 \@@_qpoint:n { col - \l_tmpa_tl }

```

```

4111 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4112 { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4113 { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4114 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4115 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4116 \clist_map_inline:Nn \l_@@_rows_tl
4117 {
4118   \tl_set:Nn \l_tmpa_tl { #####1 }
4119   \tl_if_in:NnTF \l_tmpa_tl { - }
4120   { \@@_cut_on_hyphen:w #####1 \q_stop }
4121   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4122   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4123   \tl_if_empty:NT \l_tmpb_tl
4124   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4125   \int_compare:nNnT \l_tmpb_tl > \c@iRow
4126   { \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.

```

4127 \seq_if_in:NxF \l_@@_corners_cells_seq
4128 { \l_tmpa_tl - \l_tmpc_tl }
4129 {
4130   \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
4131   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4132   \@@_qpoint:n { row - \l_tmpa_tl }
4133   \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4134   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4135   \pgfpathrectanglecorners
4136   { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4137   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4138 }
4139 }
4140 }
4141 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands \@@_rowcolors, \@@_columncolor and \@@_rowcolor:n (used in \@@_rowcolor).

```

4142 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

The following command will be used only with \l_@@_cols_tl and \c@jCol (first case) or with \l_@@_rows_tl and \c@iRow (second case). For instance, with \l_@@_cols_tl equal to 2,4-6,8-* and \c@jCol equal to 10, the clist \l_@@_cols_tl will be replaced by 2,4,5,6,8,9,10.

```

4143 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4144 {
4145   \clist_set_eq:NN \l_tmpa_clist #1
4146   \clist_clear:N #1
4147   \clist_map_inline:Nn \l_tmpa_clist
4148   {
4149     \tl_set:Nn \l_tmpa_tl { ##1 }
4150     \tl_if_in:NnTF \l_tmpa_tl { - }
4151     { \@@_cut_on_hyphen:w ##1 \q_stop }
4152     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4153     \bool_lazy_or:nnT
4154     { \tl_if_blank_p:V \l_tmpa_tl }
4155     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4156     { \tl_set:Nn \l_tmpa_tl { 1 } }
4157     \bool_lazy_or:nnT
4158     { \tl_if_blank_p:V \l_tmpb_tl }
4159     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4160     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4161     \int_compare:nNnT \l_tmpb_tl > #2
4162     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }

```

```

4163     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4164     { \clist_put_right:Nn #1 { ###1 } }
4165   }
4166 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```

4167 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4168 {
4169   \peek_remove_spaces:n
4170   {
4171     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4172     {

```

We must not expand the color (`#2`) because the color may contain the token `!` which may be activated by some packages (ex.: `babel` with the option `french` on `latex` and `pdflatex`).

```

4173       \cellcolor [ #1 ] { \exp_not:n { #2 } }
4174       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4175     }
4176   }
4177 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the tabular.

```

4178 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4179 {
4180   \peek_remove_spaces:n
4181   {
4182     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4183     {
4184       \exp_not:N \rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4185       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4186       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4187     }
4188   }
4189 }

```

```

4190 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4191 {

```

With the following line, we test whether the cell is the first one we encounter in its column (don't forget that some rows may be incomplete).

```

4192   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4193   {

```

You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells). Be careful: maybe this is not effective since we have an analyze of the instructions in the `\CodeBefore` in order to fill color by color (to avoid the thin white lines).

```

4194     \tl_gput_left:Nx \g_nicematrix_code_before_tl
4195     {
4196       \exp_not:N \columncolor [ #1 ]
4197       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4198     }
4199   }
4200 }

```

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`.

```
4201 \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`.

```
4202 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4203 {
4204   \int_compare:nNnTF \l_@@_first_col_int = 0
4205     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4206     {
4207       \int_compare:nNnTF \c@jCol = 0
4208       {
4209         \int_compare:nNnF \c@iRow = { -1 }
4210         { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4211       }
4212       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4213     }
4214 }
```

This definition may seem complicated by we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell.

The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```
4215 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4216 {
4217   \int_compare:nNnF \c@iRow = 0
4218   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4219 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the column `#1` (that is to say on the left side). `#2` is the number of consecutive occurrences of `|`.

```
4220 \cs_new_protected:Npn \@@_vline:nn #1 #2
4221 {
```

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).

```
4222   \int_compare:nNnT { #1 } < { \c@jCol + 2 }
4223   {
4224     \pgfpicture
4225     \@@_vline_i:nn { #1 } { #2 }
4226     \endpgfpicture
4227   }
```

```
4229 \cs_new_protected:Npn \@@_vline_i:nn #1 #2
4230 {
```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a row corresponding to a rule to draw, we note its number in `\l_tmpc_tl`.

```
4231   \tl_set:Nx \l_tmpb_tl { #1 }
4232   \tl_clear_new:N \l_tmpc_tl
4233   \int_step_variable:nNn \c@iRow \l_tmpa_tl
4234   {
```

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.

```

4235     \bool_gset_true:N \g_tmpa_bool
4236     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4237       { \@@_test_vline_in_block:nnnn ##1 }
4238     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4239       { \@@_test_vline_in_block:nnnn ##1 }
4240     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4241       { \@@_test_vline_in_stroken_block:nnnn ##1 }
4242     \clist_if_empty:NF \l_@@_corners_clist
4243       \@@_test_in_corner_v:
4244     \bool_if:NTF \g_tmpa_bool
4245       {
4246         \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4247         { \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl }
4248       }
4249     {
4250       \tl_if_empty:NF \l_tmpc_tl
4251       {
4252         \@@_vline_ii:nnnn
4253         { #1 }
4254         { #2 }
4255         \l_tmpc_tl
4256         { \int_eval:n { \l_tmpa_tl - 1 } }
4257         \tl_clear:N \l_tmpc_tl
4258       }
4259     }
4260   }
4261   \tl_if_empty:NF \l_tmpc_tl
4262   {
4263     \@@_vline_ii:nnnn
4264     { #1 }
4265     { #2 }
4266     \l_tmpc_tl
4267     { \int_use:N \c@iRow }
4268     \tl_clear:N \l_tmpc_tl
4269   }
4270 }

```

```

4271 \cs_new_protected:Npn \@@_test_in_corner_v:
4272 {
4273   \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
4274   {
4275     \seq_if_in:NxT
4276       \l_@@_corners_cells_seq
4277       { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4278       { \bool_set_false:N \g_tmpa_bool }
4279   }
4280   {
4281     \seq_if_in:NxT
4282       \l_@@_corners_cells_seq
4283       { \l_tmpa_tl - \l_tmpb_tl }
4284     {
4285       \int_compare:nNnTF \l_tmpb_tl = 1
4286       { \bool_set_false:N \g_tmpa_bool }
4287     }
4288     \seq_if_in:NxT
4289       \l_@@_corners_cells_seq
4290       { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }

```

```

4291         { \bool_set_false:N \g_tmpa_bool }
4292     }
4293 }
4294 }
4295 }

```

#1 is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the numbers of the rows between which the rule has to be drawn.

```

4296 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
4297 {
4298     \pgfrememberpicturepositiononpagetrue
4299     \pgf@relevantforpicturesizefalse
4300     \@@_qpoint:n { row - #3 }
4301     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4302     \@@_qpoint:n { col - #1 }
4303     \dim_set_eq:NN \l_tmpb_dim \pgf@x
4304     \@@_qpoint:n { row - \@@_succ:n { #4 } }
4305     \dim_set_eq:NN \l_tmpc_dim \pgf@y
4306     \bool_lazy_and:nnT
4307         { \int_compare_p:nNn { #2 } > 1 }
4308         { ! \tl_if_blank_p:V \CT@drsc@ }
4309     {
4310         \group_begin:
4311         \CT@drsc@
4312         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4313         \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
4314         \dim_set:Nn \l_tmpd_dim
4315             { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4316         \pgfpathrectanglecorners
4317             { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4318             { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4319         \pgfusepath { fill }
4320         \group_end:
4321     }
4322     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4323     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4324     \prg_replicate:nn { #2 - 1 }
4325     {
4326         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4327         \dim_sub:Nn \l_tmpb_dim \doublerulesep
4328         \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4329         \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4330     }
4331     \CT@arc@
4332     \pgfsetlinewidth { 1.1 \arrayrulewidth }
4333     \pgfsetrectcap
4334     \pgfusepathqstroke
4335 }

```

The following command draws a complete vertical rule in the column #1 (#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key `corners` is not used).

```

4336 \cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
4337 { \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

```

The command `\@@_draw_hlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

4338 \cs_new_protected:Npn \@@_draw_vlines:
4339 {
4340     \int_step_inline:nnn
4341     { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }

```

```

4342     { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
4343     {
4344       \tl_if_eq:NnF \l_@@_vlines_clist { all }
4345       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4346       { \@@_vline:nn { ##1 } 1 }
4347     }
4348   }

```

The horizontal rules

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the row #1. #2 is the number of consecutive occurrences of \Hline.

```

4349 \cs_new_protected:Npn \@@_hline:nn #1 #2
4350 {
4351   \pgfpicture
4352   \@@_hline_i:nn { #1 } { #2 }
4353   \endpgfpicture
4354 }
4355 \cs_new_protected:Npn \@@_hline_i:nn #1 #2
4356 {

```

\l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. When we have found a column corresponding to a rule to draw, we note its number in \l_tmpc_tl.

```

4357   \tl_set:Nn \l_tmpa_tl { #1 }
4358   \tl_clear_new:N \l_tmpc_tl
4359   \int_step_variable:nNn \c@jCol \l_tmpb_tl
4360   {

```

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.

```

4361     \bool_gset_true:N \g_tmpa_bool
4362     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4363     { \@@_test_hline_in_block:nnnn ##1 }
4364     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4365     { \@@_test_hline_in_block:nnnn ##1 }
4366     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4367     { \@@_test_hline_in_stroken_block:nnnn ##1 }
4368     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4369     \bool_if:NTF \g_tmpa_bool
4370     {
4371       \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4372       { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
4373     }
4374   {
4375     \tl_if_empty:NF \l_tmpc_tl
4376     {
4377       \@@_hline_ii:nnnn
4378       { #1 }
4379       { #2 }
4380       \l_tmpc_tl
4381       { \int_eval:n { \l_tmpb_tl - 1 } }
4382       \tl_clear:N \l_tmpc_tl
4383     }
4384   }
4385 }
4386 \tl_if_empty:NF \l_tmpc_tl
4387 {

```

```

4388 \@@_hline_ii:nnnn
4389 { #1 }
4390 { #2 }
4391 \l_tmpc_tl
4392 { \int_use:N \c@jCol }
4393 \tl_clear:N \l_tmpc_tl
4394 }
4395 }

4396 \cs_new_protected:Npn \@@_test_in_corner_h:
4397 {
4398   \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
4399   {
4400     \seq_if_in:NxT
4401       \l_@@_corners_cells_seq
4402       { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4403       { \bool_set_false:N \g_tmpa_bool }
4404   }
4405   {
4406     \seq_if_in:NxT
4407       \l_@@_corners_cells_seq
4408       { \l_tmpa_tl - \l_tmpb_tl }
4409       {
4410         \int_compare:nNnTF \l_tmpa_tl = 1
4411         { \bool_set_false:N \g_tmpa_bool }
4412         {
4413           \seq_if_in:NxT
4414             \l_@@_corners_cells_seq
4415             { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4416             { \bool_set_false:N \g_tmpa_bool }
4417         }
4418       }
4419   }
4420 }

```

#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.

```

4421 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
4422 {
4423   \pgfrememberpicturepositiononpagetrue
4424   \pgf@relevantforpicturesizefalse
4425   \@@_qpoint:n { col - #3 }
4426   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4427   \@@_qpoint:n { row - #1 }
4428   \dim_set_eq:NN \l_tmpb_dim \pgf@y
4429   \@@_qpoint:n { col - \@@_succ:n { #4 } }
4430   \dim_set_eq:NN \l_tmpc_dim \pgf@x
4431   \bool_lazy_and:nnT
4432     { \int_compare_p:nNn { #2 } > 1 }
4433     { ! \tl_if_blank_p:V \CT@drsc@ }
4434   {
4435     \group_begin:
4436     \CT@drsc@
4437     \dim_set:Nn \l_tmpd_dim
4438       { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4439     \pgfpathrectanglecorners
4440       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4441       { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4442     \pgfusepathqfill
4443     \group_end:
4444   }
4445   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```



```

4446 \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4447 \prg_replicate:nn { #2 - 1 }
4448 {
4449   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4450   \dim_sub:Nn \l_tmpb_dim \doublerulesep
4451   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4452   \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4453 }
4454 \CT@arc@
4455 \pgfsetlinewidth { 1.1 \arrayrulewidth }
4456 \pgfsetrectcap
4457 \pgfusepathqstroke
4458 }

4459 \cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
4460 { \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as `\Cdots` and in the corners (if the key `corners` is used).

```

4461 \cs_new_protected:Npn \@@_draw_hlines:
4462 {
4463   \int_step_inline:nnn
4464   { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
4465   { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
4466   {
4467     \tl_if_eq:NnF \l_@@_hlines_clist { all }
4468     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
4469     { \@@_hline:nn { ##1 } 1 }
4470   }
4471 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

4472 \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.

```

4473 \cs_set:Npn \@@_Hline_i:n #1
4474 {
4475   \peek_meaning_ignore_spaces:NTF \Hline
4476   { \@@_Hline_ii:nn { #1 + 1 } }
4477   { \@@_Hline_iii:n { #1 } }
4478 }
4479 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
4480 \cs_set:Npn \@@_Hline_iii:n #1
4481 {
4482   \skip_vertical:n
4483   {
4484     \arrayrulewidth * ( #1 )
4485     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
4486   }
4487   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4488   { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
4489   \ifnum 0 = `{ \fi }
4490 }

```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to false.

```
4491 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4
4492 {
4493   \bool_lazy_all:nT
4494   {
4495     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
4496     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4497     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4498     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4499   }
4500   { \bool_gset_false:N \g_tmpa_bool }
4501 }
```

The same for vertical rules.

```
4502 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4
4503 {
4504   \bool_lazy_all:nT
4505   {
4506     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4507     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4508     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
4509     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4510   }
4511   { \bool_gset_false:N \g_tmpa_bool }
4512 }
4513 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
4514 {
4515   \bool_lazy_all:nT
4516   {
4517     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4518     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
4519     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4520     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4521   }
4522   { \bool_gset_false:N \g_tmpa_bool }
4523 }
4524 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
4525 {
4526   \bool_lazy_all:nT
4527   {
4528     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4529     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4530     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4531     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
4532   }
4533   { \bool_gset_false:N \g_tmpa_bool }
4534 }
```

The key corners

When the key `corners` is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

```
4535 \cs_new_protected:Npn \@@_compute_corners:
4536 {
```

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

```

4537 \seq_clear_new:N \l_@@_corners_cells_seq
4538 \clist_map_inline:Nn \l_@@_corners_clist
4539 {
4540   \str_case:nnF { ##1 }
4541   {
4542     { NW }
4543     { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
4544     { NE }
4545     { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
4546     { SW }
4547     { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
4548     { SE }
4549     { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
4550   }
4551   { \@@_error:nn { bad~corner } { ##1 } }
4552 }

```

Even if the user has used the key `corners` (or the key `hvlines-except-corners`), the list of cells in the corners may be empty.

```

4553 \seq_if_empty:NF \l_@@_corners_cells_seq
4554 {

```

You write on the `aux` file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which color the `rows`, `columns` and `cells` must not color the cells in the corners.

```

4555 \iow_now:Nn \@mainaux \ExplSyntaxOn
4556 \iow_now:Nx \@mainaux
4557 {
4558   \seq_gset_from_clist:cn
4559   { c_@@_corners_cells_ \int_use:N \g_@@_env_int _ seq }
4560   { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
4561 }
4562 \iow_now:Nn \@mainaux \ExplSyntaxOff
4563 }
4564 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_corners_cells_seq`.

The six arguments of `\@@_compute_a_corner:nnnnnn` are as follow:

- **#1** and **#2** are the number of row and column of the cell which is actually in the corner;
- **#3** and **#4** are the steps in rows and the step in columns when moving from the corner;
- **#5** is the number of the final row when scanning the rows from the corner;
- **#6** is the number of the final column when scanning the columns from the corner.

```

4565 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
4566 {

```

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.

```

4567 \bool_set_false:N \l_tmpa_bool
4568 \int_zero_new:N \l_@@_last_empty_row_int
4569 \int_set:Nn \l_@@_last_empty_row_int { #1 }
4570 \int_step_inline:nnnn { #1 } { #3 } { #5 }
4571 {
4572   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }

```

```

4573 \bool_lazy_or:nnTF
4574 {
4575   \cs_if_exist_p:c
4576   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
4577 }
4578 \l_tmpb_bool
4579 { \bool_set_true:N \l_tmpa_bool }
4580 {
4581   \bool_if:NF \l_tmpa_bool
4582   { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
4583 }
4584 }

```

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

```

4585 \bool_set_false:N \l_tmpa_bool
4586 \int_zero_new:N \l_@@_last_empty_column_int
4587 \int_set:Nn \l_@@_last_empty_column_int { #2 }
4588 \int_step_inline:nnnn { #2 } { #4 } { #6 }
4589 {
4590   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
4591   \bool_lazy_or:nnTF
4592   \l_tmpb_bool
4593   {
4594     \cs_if_exist_p:c
4595     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
4596   }
4597   { \bool_set_true:N \l_tmpa_bool }
4598   {
4599     \bool_if:NF \l_tmpa_bool
4600     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
4601   }
4602 }

```

Now, we loop over the rows.

```

4603 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
4604 {

```

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

```

4605 \bool_set_false:N \l_tmpa_bool
4606 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
4607 {
4608   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
4609   \bool_lazy_or:nnTF
4610   \l_tmpb_bool
4611   {
4612     \cs_if_exist_p:c
4613     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
4614   }
4615   { \bool_set_true:N \l_tmpa_bool }
4616   {
4617     \bool_if:NF \l_tmpa_bool
4618     {
4619       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
4620       \seq_put_right:Nn
4621       \l_@@_corners_cells_seq
4622       { ##1 - #####1 }
4623     }
4624   }
4625 }
4626 }
4627 }

```

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`).

```

4628 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
4629 {
4630   \int_set:Nn \l_tmpa_int { #1 }
4631   \int_set:Nn \l_tmpb_int { #2 }
4632   \bool_set_false:N \l_tmpb_bool
4633   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4634     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
4635 }
4636 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6
4637 {
4638   \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
4639   {
4640     \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
4641     {
4642       \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
4643       {
4644         \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
4645         { \bool_set_true:N \l_tmpb_bool }
4646       }
4647     }
4648   }
4649 }

```

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`.

```

4650 \cs_new:Npn \@@_hdottedline:
4651 {
4652   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
4653   \@@_hdottedline_i:
4654 }

```

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

```

4655 \cs_new_protected:Npn \@@_hdottedline_i:
4656 {

```

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).

```

4657   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4658     { \@@_hdottedline:n { \int_use:N \c@iRow } }
4659 }

```

The command `\@@_hdottedline:n` is the command written in the `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```

4660 \AtBeginDocument
4661 {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That's why we construct now a version of `\@@_hdottedline:n` with the right environment (`\begin{pgfpicture}\end{pgfpicture}` or `\begin{tikzpicture}...\end{tikzpicture}`).

```

4662   \cs_new_protected:Npx \@@_hdottedline:n #1
4663   {
4664     \bool_set_true:N \exp_not:N \l_@@_initial_open_bool

```

```

4665     \bool_set_true:N \exp_not:N \l_@@_final_open_bool
4666     \c_@@_pgfortikzpicture_tl
4667     \@@_hdottedline_i:n { #1 }
4668     \c_@@_endpgfortikzpicture_tl
4669   }
4670 }

```

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

```

4671 \cs_new_protected:Npn \@@_hdottedline_i:n #1
4672 {
4673   \pgfrememberpicturepositiononpagetrue
4674   \@@_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).

```

4675   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4676   \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
4677   \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}
```

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

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}
```

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

```

4678   \@@_qpoint:n { col - 1 }
4679   \dim_set:Nn \l_@@_x_initial_dim
4680   {
4681     \pgf@x +

```

We do a reduction by \arraycolsep for the environments with delimiters (and not for the other).

```

4682     \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4683     - \l_@@_left_margin_dim
4684   }
4685   \@@_qpoint:n { col - \@@_succ:n \c@jCol }
4686   \dim_set:Nn \l_@@_x_final_dim
4687   {
4688     \pgf@x -
4689     \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4690     + \l_@@_right_margin_dim
4691   }

```

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.

```

4692   \tl_if_eq:NnF \g_@@_left_delim_tl (
4693     { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
4694   \tl_if_eq:NnF \g_@@_right_delim_tl )
4695   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }

```

As for now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “.” in the preamble. That’s why we impose the style `standard`.

```

4696     \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4697     \@@_draw_line:
4698 }

```

Vertical dotted lines

```

4699 \cs_new_protected:Npn \@@_vdottedline:n #1
4700 {
4701     \bool_set_true:N \l_@@_initial_open_bool
4702     \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”.

```

4703     \bool_if:NTF \c_@@_tikz_loaded_bool
4704     {
4705         \tikzpicture
4706         \@@_vdottedline_i:n { #1 }
4707         \endtikzpicture
4708     }
4709     {
4710         \pgfpicture
4711         \@@_vdottedline_i:n { #1 }
4712         \endpgfpicture
4713     }
4714 }

4715 \cs_new_protected:Npn \@@_vdottedline_i:n #1
4716 {

```

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.

```

4717     \CT@arc@
4718     \pgfrememberpicturepositiononpagetrue
4719     \@@_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).

```

4720     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
4721     \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
4722     \@@_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.

```

4723     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
4724     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
4725     \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }

```

As for now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “.” in the preamble. That’s why we impose the style `standard`.

```

4726     \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4727     \@@_draw_line:
4728 }

```

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.

```
4729 \bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```
4730 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
4731 {
4732   auto-columns-width .code:n =
4733   {
4734     \bool_set_true:N \l_@@_block_auto_columns_width_bool
4735     \dim_gzero_new:N \g_@@_max_cell_width_dim
4736     \bool_set_true:N \l_@@_auto_columns_width_bool
4737   }
4738 }

4739 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
4740 {
4741   \int_gincr:N \g_@@_NiceMatrixBlock_int
4742   \dim_zero:N \l_@@_columns_width_dim
4743   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
4744   \bool_if:NT \l_@@_block_auto_columns_width_bool
4745   {
4746     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4747     {
4748       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
4749       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4750     }
4751   }
4752 }
```

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).

```
4753 {
4754   \bool_if:NT \l_@@_block_auto_columns_width_bool
4755   {
4756     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
4757     \iow_shipout:Nx \@mainaux
4758     {
4759       \cs_gset:cpn
4760       { @@_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.

```
4761       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
4762     }
4763     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
4764   }
4765 }
```

The extra nodes

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

```
4766 \cs_generate_variant:Nn \dim_min:nn { v n }
4767 \cs_generate_variant:Nn \dim_max:nn { v n }
```

The following command is called in \@@_use_arraybox_with_notes_c: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).


```

4768 \cs_new_protected:Npn \@@_create_extra_nodes:
4769 {
4770   \bool_if:NTF \l_@@_medium_nodes_bool
4771   {
4772     \bool_if:NTF \l_@@_large_nodes_bool
4773     \@@_create_medium_and_large_nodes:
4774     \@@_create_medium_nodes:
4775   }
4776   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
4777 }

```

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`.

```

4778 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
4779 {
4780   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4781   {
4782     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
4783     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
4784     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
4785     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
4786   }
4787   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4788   {
4789     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
4790     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
4791     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
4792     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
4793   }

```

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

```

4794   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4795   {
4796     \int_step_variable:nnNn
4797     \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.

```

4798     {
4799       \cs_if_exist:cT
4800       { 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`.

```

4801     {
4802       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
4803       \dim_set:cn { l_@@_row\_@@_i: _min_dim}

```

```

4804         { \dim_min:vn { l_@@_row _ @@_i: _min_dim } \pgf@y }
4805     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { @@_i: - @@_j: }
4806     {
4807         \dim_set:cn { l_@@_column _ @@_j: _min_dim }
4808         { \dim_min:vn { l_@@_column _ @@_j: _min_dim } \pgf@x }
4809     }

```

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`.

```

4810     \pgfpointanchor { @@_env: - @@_i: - @@_j: } { north-east }
4811     \dim_set:cn { l_@@_row _ @@_i: _ max_dim }
4812     { \dim_max:vn { l_@@_row _ @@_i: _ max_dim } \pgf@y }
4813     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { @@_i: - @@_j: }
4814     {
4815         \dim_set:cn { l_@@_column _ @@_j: _ max_dim }
4816         { \dim_max:vn { l_@@_column _ @@_j: _ max_dim } \pgf@x }
4817     }
4818 }
4819 }
4820 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

4821     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int @@_i:
4822     {
4823         \dim_compare:nNnT
4824         { \dim_use:c { l_@@_row _ @@_i: _ min _ dim } } = \c_max_dim
4825         {
4826             @@_qpoint:n { row - @@_i: - base }
4827             \dim_set:cn { l_@@_row _ @@_i: _ max _ dim } \pgf@y
4828             \dim_set:cn { l_@@_row _ @@_i: _ min _ dim } \pgf@y
4829         }
4830     }
4831     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int @@_j:
4832     {
4833         \dim_compare:nNnT
4834         { \dim_use:c { l_@@_column _ @@_j: _ min _ dim } } = \c_max_dim
4835         {
4836             @@_qpoint:n { col - @@_j: }
4837             \dim_set:cn { l_@@_column _ @@_j: _ max _ dim } \pgf@y
4838             \dim_set:cn { l_@@_column _ @@_j: _ min _ dim } \pgf@y
4839         }
4840     }
4841 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

4842 \cs_new_protected:Npn \@@_create_medium_nodes:
4843 {
4844     \pgfpicture
4845     \pgfrememberpicturepositiononpagetrue
4846     \pgf@relevantforpicturesizefalse
4847     \@@_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”.

```

4848     \tl_set:Nn \l_@@_suffix_tl { -medium }
4849     \@@_create_nodes:
4850     \endpgfpicture
4851 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁶¹. 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:`.

```

4852 \cs_new_protected:Npn \@@_create_large_nodes:
4853 {
4854   \pgfpicture
4855     \pgfrememberpicturepositiononpagetrue
4856     \pgf@relevantforpicturesizefalse
4857     \@@_computations_for_medium_nodes:
4858     \@@_computations_for_large_nodes:
4859     \tl_set:Nn \l_@@_suffix_tl { - large }
4860     \@@_create_nodes:
4861   \endpgfpicture
4862 }

4863 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
4864 {
4865   \pgfpicture
4866     \pgfrememberpicturepositiononpagetrue
4867     \pgf@relevantforpicturesizefalse
4868     \@@_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”.

```

4869   \tl_set:Nn \l_@@_suffix_tl { - medium }
4870   \@@_create_nodes:
4871   \@@_computations_for_large_nodes:
4872   \tl_set:Nn \l_@@_suffix_tl { - large }
4873   \@@_create_nodes:
4874 \endpgfpicture
4875 }

```

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.

```

4876 \cs_new_protected:Npn \@@_computations_for_large_nodes:
4877 {
4878   \int_set:Nn \l_@@_first_row_int 1
4879   \int_set:Nn \l_@@_first_col_int 1

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

4880   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
4881   {
4882     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
4883     {
4884       (
4885         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
4886         \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4887       )
4888       / 2
4889     }
4890     \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4891     { l_@@_row _ \@@_i: _ min_dim }
4892   }
4893   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
4894   {
4895     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
4896     {

```

⁶¹If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

4897      (
4898      \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
4899      \dim_use:c
4900      { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4901      )
4902      / 2
4903      }
4904      \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4905      { l_@@_column _ \@@_j: _ max _ dim }
4906      }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

4907      \dim_sub:cn
4908      { l_@@_column _ 1 _ min _ dim }
4909      \l_@@_left_margin_dim
4910      \dim_add:cn
4911      { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
4912      \l_@@_right_margin_dim
4913      }

```

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).

```

4914 \cs_new_protected:Npn \@@_create_nodes:
4915 {
4916   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4917   {
4918     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4919     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

4920       \@@_pgf_rect_node:nnnnn
4921       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4922       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
4923       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
4924       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
4925       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
4926       \str_if_empty:NF \l_@@_name_str
4927       {
4928         \pgfnodealias
4929         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4930         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4931       }
4932     }
4933   }

```

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 .

```

4934   \seq_mapthread_function:NNN
4935   \g_@@_multicolumn_cells_seq
4936   \g_@@_multicolumn_sizes_seq
4937   \@@_node_for_multicolumn:nn
4938   }

```

```

4939 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
4940 {
4941   \cs_set_nopar:Npn \@@_i: { #1 }
4942   \cs_set_nopar:Npn \@@_j: { #2 }
4943 }

```

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”).

```

4944 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
4945 {
4946   \@@_extract_coords_values: #1 \q_stop
4947   \@@_pgf_rect_node:nnnnn
4948     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4949     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
4950     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
4951     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
4952     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
4953   \str_if_empty:NF \l_@@_name_str
4954   {
4955     \pgfnodealias
4956       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4957       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
4958   }
4959 }

```

The blocks

The code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass.

```

4960 \keys_define:nn { NiceMatrix / Block / FirstPass }
4961 {
4962   l .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
4963   l .value_forbidden:n = true ,
4964   r .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
4965   r .value_forbidden:n = true ,
4966   c .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
4967   c .value_forbidden:n = true ,
4968   t .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl t ,
4969   t .value_forbidden:n = true ,
4970   b .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl b ,
4971   b .value_forbidden:n = true ,
4972   color .tl_set:N = \l_@@_color_tl ,
4973   color .value_required:n = true ,
4974 }

```

The following command `\@@_Block:` will be linked to `\Block` in the environments of `nicematrix`. We define it with `\NewExpandableDocumentCommand` because it has an optional argument between `<` and `>` (for TeX instructions put before the math mode of the label and before the beginning of the small array of the block). It’s mandatory to use an expandable command.

```

4975 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
4976 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not be provided by the user, you use 1-1 (that is to say a block of only one cell).

```

4977   \peek_remove_spaces:n
4978   {
4979     \tl_if_blank:nTF { #2 }
4980       { \@@_Block_i 1-1 \q_stop }
4981       { \@@_Block_i #2 \q_stop }
4982     { #1 } { #3 } { #4 }
4983   }
4984 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
4985 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
```

Now, the arguments have been extracted: $\#1$ is i (the number of rows of the block), $\#2$ is j (the number of columns of the block), $\#3$ is the list of key-values, $\#4$ are the tokens to put before the math mode and the beginning of the small array of the block and $\#5$ is the label of the block.

```
4986 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
4987 {
```

We recall that $\#1$ and $\#2$ have been extracted from the first mandatory argument of `\Block` (which is of the syntax i - j). However, the user is allowed to omit i or j (or both). We detect that situation by replacing a missing value by 100 (it's a convention: when the block will actually be drawn these values will be detected and interpreted as *maximal possible value* according to the actual size of the array).

```
4988   \bool_lazy_or:nnTF
4989     { \tl_if_blank_p:n { #1 } }
4990     { \str_if_eq_p:nn { #1 } { * } }
4991     { \int_set:Nn \l_tmpa_int { 100 } }
4992     { \int_set:Nn \l_tmpa_int { #1 } }
4993   \bool_lazy_or:nnTF
4994     { \tl_if_blank_p:n { #2 } }
4995     { \str_if_eq_p:nn { #2 } { * } }
4996     { \int_set:Nn \l_tmpb_int { 100 } }
4997     { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
4998   \int_compare:nNnTF \l_tmpb_int = 1
4999     {
5000       \tl_if_empty:NNTF \l_@@_cell_type_tl
5001         { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5002         { \tl_set_eq:NN \l_@@_hpos_of_block_tl \l_@@_cell_type_tl }
5003     }
5004     { \tl_set:Nn \l_@@_hpos_of_block_tl c }
```

The value of `\l_@@_hpos_of_block_tl` may be modified by the keys of the command `\Block` that we will analyze now.

```
5005   \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
5006   \tl_set:Nx \l_tmpa_tl
5007   {
5008     { \int_use:N \c@iRow }
5009     { \int_use:N \c@jCol }
5010     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5011     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5012   }
```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

$\{imin\}\{jmin\}\{imax\}\{jmax\}$.

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by curryfication).

```
5013   \bool_lazy_or:nnTF
5014     { \int_compare_p:nNn { \l_tmpa_int } = 1 }
5015     { \int_compare_p:nNn { \l_tmpb_int } = 1 }
5016     { \exp_args:Nxx \@@_Block_iv:nnnnn }
5017     { \exp_args:Nxx \@@_Block_v:nnnnn }
5018   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5019 }
```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both). In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF).

```

5020 \cs_new_protected:Npn \l_@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5021 {
5022   \int_gincr:N \g_@@_block_box_int
5023   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5024   {
5025     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5026     {
5027       \@@_actually_diagbox:nnnnnn
5028       { \int_use:N \c@iRow }
5029       { \int_use:N \c@jCol }
5030       { \int_eval:n { \c@iRow + #1 - 1 } }
5031       { \int_eval:n { \c@jCol + #2 - 1 } }
5032       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5033     }
5034   }
5035   \box_gclear_new:c
5036   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
5037   \hbox_gset:cn
5038   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
5039   {

```

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with `\set@color` and not `\color_ensure_current:` because that command seems to be bugged: it doesn't work in XeLaTeX when `fontspec` is loaded.

```

5040   \tl_if_empty:NTF \l_@@_color_tl
5041   { \int_compare:nNnT { #2 } = 1 \set@color }
5042   { \color { \l_@@_color_tl } }
5043   \group_begin:
5044   \cs_set:Npn \arraystretch { 1 }
5045   \dim_set_eq:NN \extrarowheight \c_zero_dim
5046   #4

```

If the box is rotated (the key `\rotate` may be in the previous #4), the tabular used for the content of the cell will be constructed with a format `c`. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

```

5047   \bool_if:NT \g_@@_rotate_bool { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5048   \bool_if:NTF \l_@@_NiceTabular_bool
5049   {
5050     \use:x
5051     {
5052       \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5053       { @ { } \l_@@_hpos_of_block_tl @ { } }
5054     }
5055     #5
5056     \end { tabular }
5057   }
5058   {
5059     \c_math_toggle_token
5060     \use:x
5061     {
5062       \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5063       { @ { } \l_@@_hpos_of_block_tl @ { } }
5064     }
5065     #5
5066     \end { array }

```

```

5067         \c_math_toggle_token
5068     }
5069     \group_end:
5070 }
5071 \bool_if:NT \g_@@_rotate_bool
5072 {
5073     \box_grotate:cn
5074     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5075     { 90 }
5076     \bool_gset_false:N \g_@@_rotate_bool
5077 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5078 \int_compare:nNnT { #2 } = 1
5079 {
5080     \dim_gset:Nn \g_@@_blocks_wd_dim
5081     {
5082         \dim_max:nn
5083         \g_@@_blocks_wd_dim
5084         {
5085             \box_wd:c
5086             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5087         }
5088     }
5089 }

```

If we are in a mono-row block, we take into account the height and the depth of that block for the height and the depth of the row.

```

5090 \int_compare:nNnT { #1 } = 1
5091 {
5092     \dim_gset:Nn \g_@@_blocks_ht_dim
5093     {
5094         \dim_max:nn
5095         \g_@@_blocks_ht_dim
5096         {
5097             \box_ht:c
5098             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5099         }
5100     }
5101     \dim_gset:Nn \g_@@_blocks_dp_dim
5102     {
5103         \dim_max:nn
5104         \g_@@_blocks_dp_dim
5105         {
5106             \box_dp:c
5107             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5108         }
5109     }
5110 }
5111 \seq_gput_right:Nx \g_@@_blocks_seq
5112 {
5113     \l_tmpa_tl

```

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in \l_@@_hpos_of_block_tl. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of \l_@@_hpos_of_block_tl, which is fixed by the type of current column.

```

5114     { \exp_not:n { #3 } , \l_@@_hpos_of_block_tl }
5115     {
5116         \box_use_drop:c
5117         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5118     }
5119 }
5120 }

```


The following macro is for the standard case, where the block is not mono-row and not mono-column. In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array.

```

5121 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5122 {
5123   \seq_gput_right:Nx \g_@@_blocks_seq
5124   {
5125     \l_tmpa_tl
5126     { \exp_not:n { #3 } }
5127     \exp_not:n
5128     {
5129       {
5130         \bool_if:NTF \l_@@_NiceTabular_bool
5131         {
5132           \group_begin:
5133           \cs_set:Npn \arraystretch { 1 }
5134           \dim_set_eq:NN \extrarowheight \c_zero_dim
5135           #4

```

If the box is rotated (the key `\rotate` may be in the previous #4), the tabular used for the content of the cell will be constructed with a format `c`. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

```

5136       \bool_if:NT \g_@@_rotate_bool
5137       { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5138       \use:x
5139       {
5140         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5141         { @ { } \l_@@_hpos_of_block_tl @ { } }
5142       }
5143       #5
5144       \end { tabular }
5145       \group_end:
5146     }
5147   {
5148     \group_begin:
5149     \cs_set:Npn \arraystretch { 1 }
5150     \dim_set_eq:NN \extrarowheight \c_zero_dim
5151     #4
5152     \bool_if:NT \g_@@_rotate_bool
5153     { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5154     \c_math_toggle_token
5155     \use:x
5156     {
5157       \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5158       { @ { } \l_@@_hpos_of_block_tl @ { } }
5159     }
5160     #5
5161     \end { array }
5162     \c_math_toggle_token
5163     \group_end:
5164   }
5165 }
5166 }
5167 }
5168 }

```

We recall that the options of the command `\Block` are analyzed twice: first in the cell of the array and once again when the block will be put in the array *after the construction of the array* (by using PGF).

```

5169 \keys_define:nn { NiceMatrix / Block / SecondPass }

```

```

5170 {
5171   fill .tl_set:N = \l_@@_fill_tl ,
5172   fill .value_required:n = true ,
5173   draw .tl_set:N = \l_@@_draw_tl ,
5174   draw .default:n = default ,
5175   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5176   rounded-corners .default:n = 4 pt ,
5177   color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5178   color .value_required:n = true ,
5179   borders .clist_set:N = \l_@@_borders_clist ,
5180   borders .value_required:n = true ,
5181   hvlines .bool_set:N = \l_@@_hvlines_block_bool ,
5182   hvlines .default:n = true ,
5183   line-width .dim_set:N = \l_@@_line_width_dim ,
5184   line-width .value_required:n = true ,
5185   l .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
5186   l .value_forbidden:n = true ,
5187   r .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
5188   r .value_forbidden:n = true ,
5189   c .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
5190   c .value_forbidden:n = true ,
5191   t .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl t ,
5192   t .value_forbidden:n = true ,
5193   b .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl b ,
5194   b .value_forbidden:n = true ,
5195   unknown .code:n = @@_error:n { Unknown-key-for-Block }
5196 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construction of the array. We have to revert to a clean version of `\ialign` because there may be tabulars in the `\Block` instructions that will be composed now.

```

5197 \cs_new_protected:Npn \@@_draw_blocks:
5198 {
5199   \cs_set_eq:NN \ialign \@@_old_ialign:
5200   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
5201 }
5202 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5203 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5204   \int_zero_new:N \l_@@_last_row_int
5205   \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now.

```

5206   \int_compare:nNnTF { #3 } > { 99 }
5207   { \int_set_eq:NN \l_@@_last_row_int \c{iRow} }
5208   { \int_set:Nn \l_@@_last_row_int { #3 } }
5209   \int_compare:nNnTF { #4 } > { 99 }
5210   { \int_set_eq:NN \l_@@_last_col_int \c{jCol} }
5211   { \int_set:Nn \l_@@_last_col_int { #4 } }
5212   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
5213   {
5214     \int_compare:nTF
5215     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
5216     {
5217       \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }

```

```

5218         \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
5219         \group_begin:
5220         \globaldefs = 1
5221         \@@_msg_redirect_name:nn { columns~not~used } { none }
5222         \group_end:
5223     }
5224     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5225 }
5226 {
5227     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
5228     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5229     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
5230 }
5231 }
5232 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
5233 {

```

The sequence of the positions of the blocks will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

5234     \seq_gput_left:Nn \g_@@_pos_of_blocks_seq { { #1 } { #2 } { #3 } { #4 } }

```

The group is for the keys.

```

5235     \group_begin:
5236     \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
5237     \tl_if_empty:NF \l_@@_draw_tl
5238     {
5239         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5240         {
5241             \@@_stroke_block:nnn
5242             { \exp_not:n { #5 } }
5243             { #1 - #2 }
5244             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5245         }
5246         \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
5247         { { #1 } { #2 } { #3 } { #4 } }
5248     }
5249     \bool_if:NT \l_@@_hvlines_block_bool
5250     {
5251         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5252         {
5253             \@@_hvlines_block:nnn
5254             { \exp_not:n { #5 } }
5255             { #1 - #2 }
5256             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5257         }
5258     }
5259     \clist_if_empty:NF \l_@@_borders_clist
5260     {
5261         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5262         {
5263             \@@_stroke_borders_block:nnn
5264             { \exp_not:n { #5 } }
5265             { #1 - #2 }
5266             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5267         }
5268     }
5269     \tl_if_empty:NF \l_@@_fill_tl
5270     {

```

The command `\@@_extract_brackets` will extract the potential specification of color space at the beginning of `\l_@@_fill_tl` and store it in `\l_tmpa_tl` and store the color itself in `\l_tmpb_tl`.

```

5271     \exp_last_unbraced:NV \@@_extract_brackets \l_@@_fill_tl \q_stop

```

```

5272 \tl_gput_right:Nx \g_nicematrix_code_before_tl
5273 {
5274   \exp_not:N \roundedrectanglecolor
5275   [ \l_tmpa_tl ]
5276   { \exp_not:N \l_tmpb_tl }
5277   { #1 - #2 }
5278   { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5279   { \dim_use:N \l_@@_rounded_corners_dim }
5280 }
5281 }

5282 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5283 {
5284   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5285   {
5286     \@@_actually_diagbox:nnnnnn
5287     { #1 }
5288     { #2 }
5289     { \int_use:N \l_@@_last_row_int }
5290     { \int_use:N \l_@@_last_col_int }
5291     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5292   }
5293 }

5294 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
5295 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`. The latter will be used by `nicematrix` to put the label of the node. The first one won't be used explicitly.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\
& & two & \\
three & & four & five & \\
six & & seven & eight & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

5296 \pgfpicture
5297 \pgfrememberpicturepositiononpagetrue
5298 \pgf@relevantforpicturesizefalse
5299 \@@_qpoint:n { row - #1 }
5300 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5301 \@@_qpoint:n { col - #2 }
5302 \dim_set_eq:NN \l_tmpb_dim \pgf@x
5303 \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
5304 \dim_set_eq:NN \l_tmpe_dim \pgf@y
5305 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5306 \dim_set_eq:NN \l_tmpe_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.

```

5307 \begin { pgfscope }
5308 \@@pgf_rect_node:nnnnn
5309 { \@@_env: - #1 - #2 - block }
5310 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpe_dim
5311 \end { pgfscope }

```

We construct the short node.

```

5312 \dim_set_eq:NN \l_tmpb_dim \c_max_dim
5313 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5314 {

```

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.

```

5315 \cs_if_exist:cT
5316 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5317 {
5318 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5319 {
5320 \pgfpoinanchor { \@@_env: - ##1 - #2 } { west }
5321 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
5322 }
5323 }
5324 }

```

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.

```

5325 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
5326 {
5327 \@@_qpoint:n { col - #2 }
5328 \dim_set_eq:NN \l_tmpb_dim \pgf@x
5329 }
5330 \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
5331 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5332 {
5333 \cs_if_exist:cT
5334 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5335 {
5336 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5337 {
5338 \pgfpoinanchor
5339 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5340 { east }
5341 \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
5342 }
5343 }
5344 }
5345 \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
5346 {
5347 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5348 \dim_set_eq:NN \l_tmpd_dim \pgf@x
5349 }
5350 \@@pgf_rect_node:nnnnn
5351 { \@@_env: - #1 - #2 - block - short }
5352 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpe_dim

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```

5353 \bool_if:NT \l_@@_medium_nodes_bool
5354 {
5355 \@@pgf_rect_node:nnn

```

```

5356 { \@@_env: - #1 - #2 - block - medium }
5357 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
5358 {
5359     \pgfpointanchor
5360     { \@@_env:
5361         - \int_use:N \l_@@_last_row_int
5362         - \int_use:N \l_@@_last_col_int - medium
5363     }
5364     { south-east }
5365 }
5366 }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

5367 \int_compare:nNnTF { #1 } = { #3 }
5368 {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

5369     \int_compare:nNnTF { #1 } = 0
5370     { \l_@@_code_for_first_row_tl }
5371     {
5372         \int_compare:nNnT { #1 } = \l_@@_last_row_int
5373         \l_@@_code_for_last_row_tl
5374     }

```

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`.

```

5375     \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

5376     \pgfpointanchor
5377     { \@@_env: - #1 - #2 - block - short }
5378     {
5379         \str_case:Vn \l_@@_hpos_of_block_tl
5380         {
5381             c { center }
5382             l { west }
5383             r { east }
5384         }
5385     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

5386     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
5387     \pgfset { inner-sep = \c_zero_dim }
5388     \pgfnode
5389     { rectangle }
5390     {
5391         \str_case:Vn \l_@@_hpos_of_block_tl
5392         {
5393             c { base }
5394             l { base~west }
5395             r { base~east }
5396         }
5397     }
5398     { \box_use_drop:N \l_@@_cell_box } { } { }
5399 }

```

If the number of rows is different of 1, we will put the label of the block by using the short node (the label of the block has been composed in `\l_@@_cell_box`).

```

5400 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

5401     \int_compare:nNnT { #2 } = 0
5402     { \tl_set:Nn \l_@@_hpos_of_block_tl r }
5403     \bool_if:nT \g_@@_last_col_found_bool

```

```

5404     {
5405         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5406         { \tl_set:Nn \l_@@_hpos_of_block_tl 1 }
5407     }
5408     \pgftransformshift
5409     {
5410         \pgfpointanchor
5411         { \@@_env: - #1 - #2 - block - short }
5412         {
5413             \str_case:Vn \l_@@_hpos_of_block_tl
5414             {
5415                 c { center }
5416                 l { west }
5417                 r { east }
5418             }
5419         }
5420     }
5421     \pgfset { inner~sep = \c_zero_dim }
5422     \pgfnode
5423     { rectangle }
5424     {
5425         \str_case:Vn \l_@@_hpos_of_block_tl
5426         {
5427             c { center }
5428             l { west }
5429             r { east }
5430         }
5431     }
5432     { \box_use_drop:N \l_@@_cell_box } { } { }
5433 }
5434 \endpgfpicture
5435 \group_end:
5436 }

5437 \NewDocumentCommand \@@_extract_brackets { 0 { } }
5438 {
5439     \tl_set:Nn \l_tmpa_tl { #1 }
5440     \@@_store_in_tmpl_tl
5441 }
5442 \cs_new_protected:Npn \@@_store_in_tmpl_tl #1 \q_stop
5443 { \tl_set:Nn \l_tmpl_tl { #1 } }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

5444 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
5445 {
5446     \group_begin:
5447     \tl_clear:N \l_@@_draw_tl
5448     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5449     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
5450     \pgfpicture
5451     \pgfrememberpicturepositiononpagetrue
5452     \pgf@relevantforpicturesizefalse
5453     \tl_if_empty:NF \l_@@_draw_tl
5454     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

5455         \str_if_eq:VnTF \l_@@_draw_tl { default }
5456         { \CT@arc@ }
5457         { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }

```

```

5458     }
5459     \pgfsetcornersarced
5460     {
5461         \pgfpoint
5462         { \dim_use:N \l_@@_rounded_corners_dim }
5463         { \dim_use:N \l_@@_rounded_corners_dim }
5464     }
5465     \@@_cut_on_hyphen:w #2 \q_stop
5466     \bool_lazy_and:nnT
5467     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
5468     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
5469     {
5470         \@@_qpoint:n { row - \l_tmpa_tl }
5471         \dim_set:Nn \l_tmpb_dim { \pgf@y }
5472         \@@_qpoint:n { col - \l_tmpb_tl }
5473         \dim_set:Nn \l_tmpc_dim { \pgf@x }
5474         \@@_cut_on_hyphen:w #3 \q_stop
5475         \int_compare:nNnT \l_tmpa_tl > \c@iRow
5476         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
5477         \int_compare:nNnT \l_tmpb_tl > \c@jCol
5478         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5479         \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
5480         \dim_set:Nn \l_tmpa_dim { \pgf@y }
5481         \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
5482         \dim_set:Nn \l_tmpd_dim { \pgf@x }
5483         \pgfpathrectanglecorners
5484         { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5485         { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5486         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

5487     \pgfusepath { stroke }
5488 }
5489 \endpgfpicture
5490 \group_end:
5491 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

5492 \keys_define:nn { NiceMatrix / BlockStroke }
5493 {
5494     color .tl_set:N = \l_@@_draw_tl ,
5495     draw .tl_set:N = \l_@@_draw_tl ,
5496     draw .default:n = default ,
5497     line-width .dim_set:N = \l_@@_line_width_dim ,
5498     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5499     rounded-corners .default:n = 4 pt
5500 }

```

The first argument of `\@@_hvlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

5501 \cs_new_protected:Npn \@@_hvlines_block:nnn #1 #2 #3
5502 {
5503     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5504     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5505     \@@_cut_on_hyphen:w #2 \q_stop
5506     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5507     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5508     \@@_cut_on_hyphen:w #3 \q_stop
5509     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5510     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5511     \pgfpicture
5512     \pgfrememberpicturepositiononpagetrue
5513     \pgf@relevantforpicturesizefalse

```



```

5514 \CT@arc@
5515 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

First, the vertical rules.

```

5516 \@@_qpoint:n { row - \l_tmpa_tl }
5517 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5518 \@@_qpoint:n { row - \l_tmpc_tl }
5519 \dim_set_eq:NN \l_tmpb_dim \pgf@y
5520 \int_step_inline:nnn \l_tmpd_tl \l_tmpb_tl
5521 {
5522   \@@_qpoint:n { col - ##1 }
5523   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpa_dim }
5524   \pgfpathlineto { \pgfpoint \pgf@x \l_tmpb_dim }
5525   \pgfusepathqstroke
5526 }

```

Now, the horizontal rules.

```

5527 \@@_qpoint:n { col - \l_tmpb_tl }
5528 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
5529 \@@_qpoint:n { col - \l_tmpd_tl }
5530 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \arrayrulewidth }
5531 \int_step_inline:nnn \l_tmpe_tl \l_tmpe_tl
5532 {
5533   \@@_qpoint:n { row - ##1 }
5534   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5535   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5536   \pgfusepathqstroke
5537 }
5538 \endpgfpicture
5539 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax $i-j$) and the third is the last cell of the block (with the same syntax).

```

5540 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
5541 {
5542   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5543   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5544   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
5545   { \@@_error:n { borders~forbidden } }
5546   {
5547     \clist_map_inline:Nn \l_@@_borders_clist
5548     {
5549       \clist_if_in:nnF { top , bottom , left , right } { ##1 }
5550       { \@@_error:nn { bad~border } { ##1 } }
5551     }
5552     \@@_cut_on_hyphen:w #2 \q_stop
5553     \tl_set_eq:NN \l_tmpe_tl \l_tmpe_tl
5554     \tl_set_eq:NN \l_tmpe_tl \l_tmpe_tl
5555     \@@_cut_on_hyphen:w #3 \q_stop
5556     \tl_set:Nx \l_tmpe_tl { \int_eval:n { \l_tmpe_tl + 1 } }
5557     \tl_set:Nx \l_tmpe_tl { \int_eval:n { \l_tmpe_tl + 1 } }
5558     \pgfpicture
5559     \pgfrememberpicturepositiononpagetrue
5560     \pgf@relevantforpicturesizefalse
5561     \CT@arc@
5562     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
5563     \clist_if_in:NnT \l_@@_borders_clist { right }
5564     { \@@_stroke_vertical:n \l_tmpe_tl }
5565     \clist_if_in:NnT \l_@@_borders_clist { left }
5566     { \@@_stroke_vertical:n \l_tmpe_tl }
5567     \clist_if_in:NnT \l_@@_borders_clist { bottom }
5568     { \@@_stroke_horizontal:n \l_tmpe_tl }
5569     \clist_if_in:NnT \l_@@_borders_clist { top }

```

```

5570         { \@@_stroke_horizontal:n \l_tmpc_tl }
5571     \endpgfpicture
5572 }
5573 }

```

The following command is used to stroke the left border and the right border. The argument #1 is the number of column (in the sense of the col node).

```

5574 \cs_new_protected:Npn \@@_stroke_vertical:n #1
5575 {
5576     \@@_qpoint:n \l_tmpc_tl
5577     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
5578     \@@_qpoint:n \l_tmpa_tl
5579     \dim_set:Nn \l_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
5580     \@@_qpoint:n { #1 }
5581     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
5582     \pgfpathlineto { \pgfpoint \pgf@x \l_tmpc_dim }
5583     \pgfusepathqstroke
5584 }

```

The following command is used to stroke the top border and the bottom border. The argument #1 is the number of row (in the sense of the row node).

```

5585 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
5586 {
5587     \@@_qpoint:n \l_tmpd_tl
5588     \clist_if_in:NnTF \l_@@_borders_clist { left }
5589         { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
5590         { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
5591     \@@_qpoint:n \l_tmpb_tl
5592     \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
5593     \@@_qpoint:n { #1 }
5594     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5595     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5596     \pgfusepathqstroke
5597 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

5598 \keys_define:nn { NiceMatrix / BlockBorders }
5599 {
5600     borders .clist_set:N = \l_@@_borders_clist ,
5601     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5602     rounded-corners .default:n = 4 pt ,
5603     line-width .dim_set:N = \l_@@_line_width_dim
5604 }

```

How to draw the dotted lines transparently

```

5605 \cs_set_protected:Npn \@@_renew_matrix:
5606 {
5607     \RenewDocumentEnvironment { pmatrix } { } {
5608         { \pNiceMatrix }
5609         { \endpNiceMatrix }
5610     \RenewDocumentEnvironment { vmatrix } { } {
5611         { \vNiceMatrix }
5612         { \endvNiceMatrix }
5613     \RenewDocumentEnvironment { Vmatrix } { } {
5614         { \VNiceMatrix }
5615         { \endVNiceMatrix }
5616     \RenewDocumentEnvironment { bmatrix } { } {
5617         { \bNiceMatrix }
5618         { \endbNiceMatrix }
5619     \RenewDocumentEnvironment { Bmatrix } { } {
5620         { \BNiceMatrix }

```

```

5621 { \endBNiceMatrix }
5622 }

```

Automatic arrays

```

5623 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
5624 {
5625   \int_set:Nn \l_@@_nb_rows_int { #1 }
5626   \int_set:Nn \l_@@_nb_cols_int { #2 }
5627 }
5628 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
5629 {
5630   \int_zero_new:N \l_@@_nb_rows_int
5631   \int_zero_new:N \l_@@_nb_cols_int
5632   \@@_set_size:n #4 \q_stop
5633   \begin { NiceArrayWithDelims } { #1 } { #2 }
5634     { * { \l_@@_nb_cols_int } { c } } [ #3 , #5 , #7 ]
5635   \int_compare:nNnT \l_@@_first_row_int = 0
5636     {
5637       \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5638       \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5639       \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5640     }
5641   \prg_replicate:nn \l_@@_nb_rows_int
5642     {
5643       \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`).

```

5644   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
5645   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5646 }
5647 \int_compare:nNnT \l_@@_last_row_int > { -2 }
5648 {
5649   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5650   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5651   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5652 }
5653 \end { NiceArrayWithDelims }
5654 }
5655 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
5656 {
5657   \cs_set_protected:cpn { #1 AutoNiceMatrix }
5658   {
5659     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
5660     \AutoNiceMatrixWithDelims { #2 } { #3 }
5661   }
5662 }
5663 \@@_define_com:nnn p ( )
5664 \@@_define_com:nnn b [ ]
5665 \@@_define_com:nnn v | |
5666 \@@_define_com:nnn V \ | \ |
5667 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

5668 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
5669 {
5670   \group_begin:
5671   \bool_set_true:N \l_@@_NiceArray_bool
5672   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
5673   \group_end:

```

```
5674 }
```

The redefinition of the command `\dotfill`

```
5675 \cs_set_eq:NN \@@_old_dotfill \dotfill
5676 \cs_new_protected:Npn \@@_dotfill:
5677 {
```

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}`).

```
5678   \@@_old_dotfill
5679   \bool_if:NT \l_@@_NiceTabular_bool
5680     { \group_insert_after:N \@@_dotfill_ii: }
5681     { \group_insert_after:N \@@_dotfill_i: }
5682   }
5683 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
5684 \cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }
```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```
5685 \cs_new_protected:Npn \@@_dotfill_iii:
5686 { \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`.

```
5687 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
5688 {
5689   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5690   {
5691     \@@_actually_diagbox:nnnnnn
5692     { \int_use:N \c@iRow }
5693     { \int_use:N \c@jCol }
5694     { \int_use:N \c@iRow }
5695     { \int_use:N \c@jCol }
5696     { \exp_not:n { #1 } }
5697     { \exp_not:n { #2 } }
5698   }
```

We put the cell with `\diagbox` in the sequence `\g_@@_pos_of_blocks_seq` because a cell with `\diagbox` must be considered as non empty by the key `corners`.

```
5699   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
5700   {
5701     { \int_use:N \c@iRow }
5702     { \int_use:N \c@jCol }
5703     { \int_use:N \c@iRow }
5704     { \int_use:N \c@jCol }
5705   }
5706 }
```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it’s possible to use `\diagbox` in a `\Block`). The two other are the elements to draw below and above the diagonal line.

```
5707 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
5708 {
5709   \pgfpicture
5710   \pgf@relevantforpicturesizefalse
5711   \pgfrememberpicturepositiononpagetrue
5712   \@@_qpoint:n { row - #1 }
5713   \dim_set_eq:NN \l_tmpa_dim \pgf@y
```

```

5714 \@@_qpoint:n { col - #2 }
5715 \dim_set_eq:NN \l_tmpb_dim \pgf@x
5716 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5717 \@@_qpoint:n { row - \@@_succ:n { #3 } }
5718 \dim_set_eq:NN \l_tmpc_dim \pgf@y
5719 \@@_qpoint:n { col - \@@_succ:n { #4 } }
5720 \dim_set_eq:NN \l_tmpd_dim \pgf@x
5721 \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
5722 {

```

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.

```

5723 \CT@arc@
5724 \pgfsetroundcap
5725 \pgfusepathqstroke
5726 }
5727 \pgfset { inner~sep = 1 pt }
5728 \pgfscope
5729 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
5730 \pgfnode { rectangle } { south~west }
5731 { \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }
5732 \endpgfscope
5733 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5734 \pgfnode { rectangle } { north~east }
5735 { \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
5736 \endpgfpicture
5737 }

```

The keyword `\CodeAfter`

The `\CodeAfter` (inserted with the key `code-after` or after the keyword `\CodeAfter`) may always begin with a list of pairs *key-value* between square brackets. Here is the corresponding set of keys.

```

5738 \keys_define:nn { NiceMatrix }
5739 {
5740   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
5741   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
5742 }
5743 \keys_define:nn { NiceMatrix / CodeAfter }
5744 {
5745   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
5746   sub-matrix .value_required:n = true ,
5747   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
5748   delimiters / color .value_required:n = true ,
5749   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
5750   rules .value_required:n = true ,
5751   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
5752 }

```

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. 102.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:.` That macro must *not* be protected since it begins with `\omit`.

```

5753 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_i:n }

```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_i:n` which do *not* begin with `\omit` (and thus, the user will be able to use `\CodeAfter` without error and without the need to prefix by `\omit`).

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

5754 \cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
5755 {
5756   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
5757   \@@_CodeAfter_ii:n
5758 }

```

We catch the argument of the command `\end` (in `#1`).

```

5759 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1
5760 {

```

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.

```

5761   \str_if_eq:eeTF \currenenv { #1 } { \end { #1 } }

```

If this is not the `\end` we are looking for, we put those tokens in `\g_nicematrix_code_after_tl` and we go on searching for the next command `\end` with a recursive call to the command `\@@_CodeAfter:n`.

```

5762   {
5763     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
5764     \@@_CodeAfter_i:n
5765   }
5766 }

```

The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_internal_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```

5767 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
5768 {
5769   \pgfpicture
5770   \pgfrememberpicturepositiononpagetrue
5771   \pgf@relevantforpicturesizefalse

```

`\l_@@_y_initial_dim` and `\l_@@_y_final_dim` will be the y -values of the extremities of the delimiter we will have to construct.

```

5772   \@@_qpoint:n { row - 1 }
5773   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5774   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
5775   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y

```

We will compute in `\l_tmpa_dim` the x -value where we will have to put our delimiter (on the left side or on the right side).

```

5776   \bool_if:nTF { #3 }
5777   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
5778   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
5779   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5780   {
5781     \cs_if_exist:cT
5782     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5783     {
5784       \pgfpointanchor
5785       { \@@_env: - ##1 - #2 }
5786       { \bool_if:nTF { #3 } { west } { east } }
5787       \dim_set:Nn \l_tmpa_dim
5788       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }

```

```

5789     }
5790 }

```

Now we can put the delimiter with a node of PGF.

```

5791 \pgfset { inner~sep = \c_zero_dim }
5792 \dim_zero:N \nulldelimiterspace
5793 \pgftransformshift
5794 {
5795   \pgfpoint
5796   { \l_tmpa_dim }
5797   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
5798 }
5799 \pgfnode
5800 { rectangle }
5801 { \bool_if:nTF { #3 } { east } { west } }
5802 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

5803   \nullfont
5804   \c_math_toggle_token
5805   \tl_if_empty:NF \l_@@_delimiters_color_tl
5806   { \color { \l_@@_delimiters_color_tl } }
5807   \bool_if:nTF { #3 } { \left #1 } { \left . }
5808   \vcenter
5809   {
5810     \nullfont
5811     \hrule \@height
5812       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
5813       \@depth \c_zero_dim
5814       \@width \c_zero_dim
5815   }
5816   \bool_if:nTF { #3 } { \right . } { \right #1 }
5817   \c_math_toggle_token
5818 }
5819 { }
5820 { }
5821 \endpgfpicture
5822 }

```

The command `\SubMatrix`

```

5823 \keys_define:nn { NiceMatrix / sub-matrix }
5824 {
5825   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
5826   extra-height .value_required:n = true ,
5827   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
5828   left-xshift .value_required:n = true ,
5829   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
5830   right-xshift .value_required:n = true ,
5831   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
5832   xshift .value_required:n = true ,
5833   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
5834   delimiters / color .value_required:n = true ,
5835   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
5836   slim .default:n = true ,
5837   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
5838   hlines .default:n = all ,
5839   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
5840   vlines .default:n = all ,
5841   hvlines .meta:n = { hlines, vlines } ,
5842   hvlines .value_forbidden:n = true ,
5843 }
5844 \keys_define:nn { NiceMatrix }
5845 {

```

```

5846 SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
5847 CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5848 NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5849 NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5850 pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5851 NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5852 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

5853 \keys_define:nn { NiceMatrix / SubMatrix }
5854 {
5855   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
5856   hlines .default:n = all ,
5857   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
5858   vlines .default:n = all ,
5859   hvlines .meta:n = { hlines, vlines } ,
5860   hvlines .value_forbidden:n = true ,
5861   name .code:n =
5862     \tl_if_empty:nTF { #1 }
5863     { \@@_error:n { Invalid-name-format } }
5864     {
5865       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
5866       {
5867         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
5868         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
5869         {
5870           \str_set:Nn \l_@@_submatrix_name_str { #1 }
5871           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
5872         }
5873       }
5874       { \@@_error:n { Invalid-name-format } }
5875     } ,
5876   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
5877   rules .value_required:n = true ,
5878   code .tl_set:N = \l_@@_code_tl ,
5879   code .value_required:n = true ,
5880   name .value_required:n = true ,
5881   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
5882 }

5883 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
5884 {
5885   \@@_cut_on_hyphen:w #3 \q_stop
5886   \tl_clear_new:N \l_tmpc_tl
5887   \tl_clear_new:N \l_tmpd_tl
5888   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5889   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5890   \@@_cut_on_hyphen:w #2 \q_stop
5891   \seq_gput_right:Nx \g_@@_submatrix_seq
5892   { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
5893   \tl_gput_right:Nn \g_@@_internal_code_after_tl
5894   { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
5895 }

```

In the internal code-after and in the `\CodeAfter` the following command `\@@_SubMatrix` will be linked to `\SubMatrix`.

- #1 is the left delimiter;
- #2 is the upper-left cell of the matrix with the format $i-j$;
- #3 is the lower-right cell of the matrix with the format $i-j$;

- #4 is the right delimiter;
- #5 is the list of options of the command.

```

5896 \NewDocumentCommand \@@_SubMatrix { m m m m O { } }
5897 {
5898   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

5899   \tl_clear_new:N \l_@@_first_i_tl
5900   \tl_clear_new:N \l_@@_first_j_tl
5901   \tl_clear_new:N \l_@@_last_i_tl
5902   \tl_clear_new:N \l_@@_last_j_tl

```

The command `\@@_cut_on_hyphen:w` cuts on the hyphen an argument of the form $i-j$. The value of i is stored in `\l_tmpa_tl` and the value of j is stored in `\l_tmpb_tl`.

```

5903   \@@_cut_on_hyphen:w #2 \q_stop
5904   \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
5905   \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
5906   \@@_cut_on_hyphen:w #3 \q_stop
5907   \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
5908   \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
5909   \bool_lazy_or:nnTF
5910     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
5911     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
5912     { \@@_error:n { SubMatrix~too~large } }
5913   {
5914     \str_clear_new:N \l_@@_submatrix_name_str
5915     \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
5916     \pgfpicture
5917     \pgfrememberpicturepositiononpagetrue
5918     \pgf@relevantforpicturesizefalse
5919     \pgfset { inner~sep = \c_zero_dim }
5920     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
5921     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currying.

```

5922   \bool_if:NTF \l_@@_submatrix_slim_bool
5923     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
5924     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
5925     {
5926       \cs_if_exist:cT
5927         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
5928       {
5929         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
5930         \dim_set:Nn \l_@@_x_initial_dim
5931           { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
5932       }
5933       \cs_if_exist:cT
5934         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
5935       {
5936         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
5937         \dim_set:Nn \l_@@_x_final_dim
5938           { \dim_max:nn \l_@@_x_final_dim \pgf@x }
5939       }
5940     }
5941   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
5942     { \@@_error:nn { impossible~delimiter } { left } }
5943     {
5944       \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
5945         { \@@_error:nn { impossible~delimiter } { right } }
5946         { \@@_sub_matrix_i:nn { #1 } { #4 } }
5947     }
5948   \endpgfpicture
5949 }

```

```

5950 \group_end:
5951 }

```

#1 is the left delimiter dans #2 is the right one.

```

5952 \cs_new_protected:Npn \l_@@_sub_matrix_i:nn #1 #2
5953 {
5954   \l_@@_qpoint:n { row - \l_@@_first_i_tl - base }
5955   \dim_set:Nn \l_@@_y_initial_dim
5956     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
5957   \l_@@_qpoint:n { row - \l_@@_last_i_tl - base }
5958   \dim_set:Nn \l_@@_y_final_dim
5959     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
5960   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
5961     {
5962       \cs_if_exist:cT
5963       { \pgf @ sh @ ns @ \l_@@_env: - \l_@@_first_i_tl - ##1 }
5964       {
5965         \pgfpointanchor { \l_@@_env: - \l_@@_first_i_tl - ##1 } { north }
5966         \dim_set:Nn \l_@@_y_initial_dim
5967           { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
5968       }
5969       \cs_if_exist:cT
5970       { \pgf @ sh @ ns @ \l_@@_env: - \l_@@_last_i_tl - ##1 }
5971       {
5972         \pgfpointanchor { \l_@@_env: - \l_@@_last_i_tl - ##1 } { south }
5973         \dim_set:Nn \l_@@_y_final_dim
5974           { \dim_min:nn \l_@@_y_final_dim \pgf@y }
5975       }
5976     }
5977   \dim_set:Nn \l_tmpa_dim
5978     {
5979     \l_@@_y_initial_dim - \l_@@_y_final_dim +
5980     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
5981     }
5982   \dim_set_eq:NN \nulldelimiterspace \c_zero_dim

```

We will draw the rules in the \SubMatrix.

```

5983 \group_begin:
5984 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5985 \tl_if_empty:NF \l_@@_rules_color_tl
5986 { \exp_after:wN \l_@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
5987 \CT@arc@

```

Now, we draw the potential vertical rules specified in the preamble of the environments with the letter fixed with the key `vlines-in-sub-matrix`. The list of the columns where there is such rule to draw is in `\g_@@_cols_vlism_seq`.

```

5988 \seq_map_inline:Nn \g_@@_cols_vlism_seq
5989 {
5990   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
5991   {
5992     \int_compare:nNnT
5993     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
5994     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

5995       \l_@@_qpoint:n { col - ##1 }
5996       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
5997       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
5998       \pgfusepathqstroke
5999     }
6000   }
6001 }

```

Now, we draw the vertical rules specified in the key `vlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

6002   \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6003   { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6004   { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6005   {
6006     \bool_lazy_and:nnTF
6007     { \int_compare_p:nNn { ##1 } > 0 }
6008     {
6009       \int_compare_p:nNn
6010       { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6011     {
6012       @@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6013       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6014       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6015       \pgfusepathqstroke
6016     }
6017     { @@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
6018   }

```

Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

6019   \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6020   { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6021   { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
6022   {
6023     \bool_lazy_and:nnTF
6024     { \int_compare_p:nNn { ##1 } > 0 }
6025     {
6026       \int_compare_p:nNn
6027       { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6028     {
6029       @@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

6030   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

6031   \dim_set:Nn \l_tmpa_dim
6032   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6033   \str_case:nn { #1 }
6034   {
6035     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6036     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
6037     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6038   }
6039   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

6040   \dim_set:Nn \l_tmpb_dim
6041   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6042   \str_case:nn { #2 }
6043   {
6044     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6045     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
6046     \{ { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6047   }
6048   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6049   \pgfusepathqstroke
6050   \group_end:
6051 }
6052 { @@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
6053 }

```

If the key `name` has been used for the command `\SubMatrix`, we create a PGF node with that name for the submatrix (this node does not encompass the delimiters that we will put after).

```

6054 \str_if_empty:NF \l_@@_submatrix_name_str
6055 {
6056   \pgf_rect_node:nnnnn \l_@@_submatrix_name_str
6057   \l_@@_x_initial_dim \l_@@_y_initial_dim
6058   \l_@@_x_final_dim \l_@@_y_final_dim
6059 }
6060 \group_end:

```

The group was for `\CT@arc@` (the color of the rules).

Now, we deal with the left delimiter. Of course, the environment `{pgfscope}` is for the `\pgftransformshift`.

```

6061 \begin { pgfscope }
6062 \pgftransformshift
6063 {
6064   \pgfpoint
6065   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6066   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6067 }
6068 \str_if_empty:NTF \l_@@_submatrix_name_str
6069 { \@@_node_left:nn #1 { } }
6070 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
6071 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

6072 \pgftransformshift
6073 {
6074   \pgfpoint
6075   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6076   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6077 }
6078 \str_if_empty:NTF \l_@@_submatrix_name_str
6079 { \@@_node_right:nn #2 { } }
6080 {
6081   \@@_node_right:nn #2 { \@@_env: - \l_@@_submatrix_name_str - right }
6082 }
6083 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
6084 \flag_clear_new:n { nicematrix }
6085 \l_@@_code_tl
6086 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

6087 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

6088 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
6089 {
6090   \use:e
6091   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpointanchor_i:nn #1 } }
6092 }

```

In fact, the argument of `\pgfpointanchor` is always of the form `\a_command { name_of_node }` where “name_of_node” is the name of the Tikz node without the potential prefix and suffix. That’s why we catch two arguments and work only on the second by trying (first) to extract an hyphen -.

```
6093 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
6094   { #1 { \@@_pgfpointanchor_ii:w #2 - \q_stop } }
```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```
6095 \tl_const:Nn \c_@@_integers_alist_tl
6096   {
6097     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
6098     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
6099     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
6100     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
6101   }
```

```
6102 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
6103   {
```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form i - $|j$. In that case, the i of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the j arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row of a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```
6104   \tl_if_empty:nTF { #2 }
6105   {
6106     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
6107     {
6108       \flag_raise:n { nicematrix }
6109       \int_if_even:nTF { \flag_height:n { nicematrix } }
6110       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
6111       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
6112     }
6113     { #1 }
6114   }
```

If there is an hyphen, we have to see whether we have a node of the form i - j , row- i or col- j .

```
6115   { \@@_pgfpointanchor_iii:w { #1 } #2 }
6116 }
```

There was an hyphen in the name of the node and that’s why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```
6117 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
6118   {
6119     \str_case:nnF { #1 }
6120     {
6121       { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
6122       { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
6123     }
6124   }
```

Now the case of a node of the form i - j .

```
6124   {
6125     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
6126     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
6127   }
6128 }
```

The command `\@@_node_left:nn` puts the left delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

6129 \cs_new_protected:Npn \@@_node_left:nn #1 #2
6130 {
6131   \pgfnode
6132     { rectangle }
6133     { east }
6134     {
6135       \nullfont
6136       \c_math_toggle_token
6137       \tl_if_empty:NF \l_@@_delimiters_color_tl
6138       { \color { \l_@@_delimiters_color_tl } }
6139       \left #1
6140       \vcenter
6141       {
6142         \nullfont
6143         \hrule \@height \l_tmpa_dim
6144           \@depth \c_zero_dim
6145           \@width \c_zero_dim
6146       }
6147       \right .
6148       \c_math_toggle_token
6149     }
6150     { #2 }
6151     { }
6152 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

6153 \cs_new_protected:Npn \@@_node_right:nn #1 #2
6154 {
6155   \pgfnode
6156     { rectangle }
6157     { west }
6158     {
6159       \nullfont
6160       \c_math_toggle_token
6161       \tl_if_empty:NF \l_@@_delimiters_color_tl
6162       { \color { \l_@@_delimiters_color_tl } }
6163       \left .
6164       \vcenter
6165       {
6166         \nullfont
6167         \hrule \@height \l_tmpa_dim
6168           \@depth \c_zero_dim
6169           \@width \c_zero_dim
6170       }
6171       \right #1
6172       \c_math_toggle_token
6173     }
6174     { #2 }
6175     { }
6176 }

```

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.

```
6177 \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.

```
6178 \bool_new:N \c_@@_footnote_bool
6179 \@@_msg_new:nnn { Unknown~option~for~package }
6180 {
6181   The~key~'\l_keys_key_str'~is~unknown. \\
6182   If~you~go~on,~it~will~be~ignored. \\
6183   For~a~list~of~the~available~keys,~type~H~<return>.
6184 }
6185 {
6186   The~available~keys~are~(in~alphabetic~order):~
6187   define-L-C-R,~
6188   footnote,~
6189   footnotehyper,~
6190   renew-dots,~and
6191   renew-matrix.
6192 }
```

Maybe we will completely delete the key 'transparent' in a future version.

```
6193 \@@_msg_new:nn { Key~transparent }
6194 {
6195   The~key~'transparent'~is~now~obsolete~(because~it's~name~
6196   is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
6197   and~'renew-matrix'.~However,~you~can~go~on.
6198 }
6199 \keys_define:nn { NiceMatrix / Package }
6200 {
6201   define-L-C-R .bool_set:N = \c_@@_define_L_C_R_bool ,
6202   define-L-C-R .default:n = true ,
6203   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
6204   renew-dots .value_forbidden:n = true ,
6205   renew-matrix .code:n = \@@_renew_matrix: ,
6206   renew-matrix .value_forbidden:n = true ,
6207   transparent .code:n =
6208     {
6209       \@@_renew_matrix:
6210       \bool_set_true:N \l_@@_renew_dots_bool
6211       \@@_error:n { Key~transparent }
6212     } ,
6213   transparent .value_forbidden:n = true,
6214   footnote .bool_set:N = \c_@@_footnote_bool ,
6215   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
6216   unknown .code:n = \@@_error:n { Unknown~option~for~package }
6217 }
6218 \ProcessKeysOptions { NiceMatrix / Package }
```

```
6219 \@@_msg_new:nn { footnote~with~footnotehyper~package }
6220 {
6221   You~can't~use~the~option~'footnote'~because~the~package~
6222   footnotehyper~has~already~been~loaded.~
6223   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
6224   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6225   of~the~package~footnotehyper.\\
6226   If~you~go~on,~the~package~footnote~won't~be~loaded.
6227 }
```

```

6228 \@@_msg_new:nn { footnotehyper~with~footnote~package }
6229 {
6230   You~can't~use~the~option~'footnotehyper'~because~the~package~
6231   footnote~has~already~been~loaded.~
6232   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
6233   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6234   of~the~package~footnote.\\
6235   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
6236 }

```

```

6237 \bool_if:NT \c_@@_footnote_bool
6238 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6239   \@ifclassloaded { beamer }
6240   { \bool_set_false:N \c_@@_footnote_bool }
6241   {
6242     \@ifpackageloaded { footnotehyper }
6243     { \@_error:n { footnote~with~footnotehyper~package } }
6244     { \usepackage { footnote } }
6245   }
6246 }

6247 \bool_if:NT \c_@@_footnotehyper_bool
6248 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6249   \@ifclassloaded { beamer }
6250   { \bool_set_false:N \c_@@_footnote_bool }
6251   {
6252     \@ifpackageloaded { footnote }
6253     { \@_error:n { footnotehyper~with~footnote~package } }
6254     { \usepackage { footnotehyper } }
6255   }
6256   \bool_set_true:N \c_@@_footnote_bool
6257 }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

Error messages of the package

The following message will be deleted when we will delete the key `except-corners` for the command `\arraycolor`.

```

6258 \@@_msg_new:nn { key except-corners }
6259 {
6260   The~key~'except-corners'~has~been~deleted~for~the~command~\token_to_str:N
6261   \arraycolor\ in~the~\token_to_str:N \CodeBefore.~You~should~instead~use~
6262   the~key~'corners'~in~your~\@@_full_name_env:.\\
6263   If~you~go~on,~this~key~will~be~ignored.
6264 }

6265 \seq_new:N \c_@@_types_of_matrix_seq
6266 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
6267 {
6268   NiceMatrix ,
6269   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
6270 }
6271 \seq_set_map_x:NNn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
6272 { \tl_to_str:n { #1 } }

```


If the user uses too much columns, the command `\@@_error_too_much_cols:` is executed. This command raises an error but try to give the best information to the user in the error message. The command `\seq_if_in:NVTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

6273 \cs_new_protected:Npn \@@_error_too_much_cols:
6274 {
6275   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
6276   {
6277     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
6278     { \@@_fatal:n { too-much-cols-for-matrix } }
6279     {
6280       \bool_if:NF \l_@@_last_col_without_value_bool
6281       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
6282     }
6283   }
6284   { \@@_fatal:n { too-much-cols-for-array } }
6285 }

```

The following command must *not* be protected since it's used in an error message.

```

6286 \cs_new:Npn \@@_message_hdotsfor:
6287 {
6288   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
6289   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is-incorrect.}
6290 }
6291 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
6292 {
6293   You-try-to-use-more-columns-than-allowed-by-your~
6294   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of~
6295   columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus-the~
6296   exterior-columns).~This-error-is-fatal.
6297 }
6298 \@@_msg_new:nn { too-much-cols-for-matrix }
6299 {
6300   You-try-to-use-more-columns-than-allowed-by-your~
6301   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
6302   number-of-columns-for-a-matrix-is-fixed-by-the-LaTeX-counter~
6303   'MaxMatrixCols'.~Its-actual-value-is~\int_use:N \c@MaxMatrixCols.~
6304   This-error-is-fatal.
6305 }

```

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`.

```

6306 \@@_msg_new:nn { too-much-cols-for-array }
6307 {
6308   You-try-to-use-more-columns-than-allowed-by-your~
6309   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
6310   \int_use:N \g_@@_static_num_of_col_int\
6311   ~ (plus-the-potential-exterior-ones).~
6312   This-error-is-fatal.
6313 }
6314 \@@_msg_new:nn { last-col-not-used }
6315 {
6316   The-key~'last-col'~is~in-force-but~you-have-not-used-that-last-column~
6317   in-your~\@@_full_name_env:~.~However,~you-can-go-on.
6318 }
6319 \@@_msg_new:nn { columns-not-used }
6320 {
6321   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
6322   \g_@@_static_num_of_col_int\ columns-but~you-use-only~\int_use:N \c@jCol.\
6323   You-can-go-on-but-the-columns-you-did-not-used-won't-be-created.
6324 }

```

```

6325 \@@_msg_new:nn { in-first-col }
6326 {
6327     You~can't~use~the~command~#1~in~the~first~column~(number~0)~of~the~array.\\
6328     If~you~go~on,~this~command~will~be~ignored.
6329 }
6330 \@@_msg_new:nn { in-last-col }
6331 {
6332     You~can't~use~the~command~#1~in~the~last~column~(exterior)~of~the~array.\\
6333     If~you~go~on,~this~command~will~be~ignored.
6334 }
6335 \@@_msg_new:nn { in-first-row }
6336 {
6337     You~can't~use~the~command~#1~in~the~first~row~(number~0)~of~the~array.\\
6338     If~you~go~on,~this~command~will~be~ignored.
6339 }
6340 \@@_msg_new:nn { in-last-row }
6341 {
6342     You~can't~use~the~command~#1~in~the~last~row~(exterior)~of~the~array.\\
6343     If~you~go~on,~this~command~will~be~ignored.
6344 }
6345 \@@_msg_new:nn { double-closing-delimiter }
6346 {
6347     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
6348     delimiter.~This~delimiter~will~be~ignored.
6349 }
6350 \@@_msg_new:nn { delimiter~after~opening }
6351 {
6352     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
6353     delimiter.~This~delimiter~will~be~ignored.
6354 }
6355 \@@_msg_new:nn { bad-option-for~line-style }
6356 {
6357     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
6358     is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
6359 }
6360 \@@_msg_new:nn { Unknown~key~for~xdots }
6361 {
6362     As~for~now,~there~is~only~three~key~available~here:~'color',~'line-style'~
6363     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
6364     this~key~will~be~ignored.
6365 }
6366 \@@_msg_new:nn { Unknown~key~for~rowcolors }
6367 {
6368     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
6369     (and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
6370     this~key~will~be~ignored.
6371 }
6372 \@@_msg_new:nn { ampersand~in~light-syntax }
6373 {
6374     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
6375     ~you~have~used~the~key~'light-syntax'.~This~error~is~fatal.
6376 }
6377 \@@_msg_new:nn { SubMatrix~too~large }
6378 {
6379     Your~command~\token_to_str:N \SubMatrix\
6380     can't~be~drawn~because~your~matrix~is~too~small.\\
6381     If~you~go~on,~this~command~will~be~ignored.
6382 }

```

```

6383 \@@_msg_new:nn { double-backslash-in-light-syntax }
6384 {
6385     You~can't~use~\token_to_str:N \~to~separate~rows~because~you~have~used~
6386     the~key~'light-syntax'.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
6387     (set~by~the~key~'end-of-row').~This~error~is~fatal.
6388 }
6389 \@@_msg_new:nn { standard-cline-in-document }
6390 {
6391     The~key~'standard-cline'~is~available~only~in~the~preamble.\\
6392     If~you~go~on~this~command~will~be~ignored.
6393 }
6394 \@@_msg_new:nn { old-column-type }
6395 {
6396     The~column~type~'#1'~is~no~longer~defined~in~'nicematrix'.~
6397     Since~version~5.0,~you~have~to~use~'l',~'c'~and~'r'~instead~of~'L',~
6398     'C'~and~'R'.~You~can~also~use~the~key~'define-L-C-R'.\\
6399     This~error~is~fatal.
6400 }
6401 \@@_msg_new:nn { bad-value-for-baseline }
6402 {
6403     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
6404     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
6405     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
6406     If~you~go~on,~a~value~of~1~will~be~used.
6407 }
6408 \@@_msg_new:nn { Invalid-name-format }
6409 {
6410     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
6411     \SubMatrix.\\
6412     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
6413     If~you~go~on,~this~key~will~be~ignored.
6414 }
6415 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
6416 {
6417     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
6418     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
6419     number~is~not~valid.~If~you~go~on,~it~will~be~ignored.
6420 }
6421 \@@_msg_new:nn { impossible-delimiter }
6422 {
6423     It's~impossible~to~draw~the~#1~delimiter~of~your~
6424     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
6425     in~that~column.
6426     \bool_if:NT \l_@@_submatrix_slim_bool
6427     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
6428     If~you~go~on,~this~\token_to_str:N \SubMatrix\ will~be~ignored.
6429 }
6430 \@@_msg_new:nn { empty-environment }
6431 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }
6432 \@@_msg_new:nn { Delimiter-with-small }
6433 {
6434     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
6435     because~the~key~'small'~is~in~force.\\
6436     This~error~is~fatal.
6437 }
6438 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
6439 {
6440     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code~after'~
6441     can't~be~executed~because~a~cell~doesn't~exist.\\
6442     If~you~go~on~this~command~will~be~ignored.
6443 }

```

```

6444 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
6445 {
6446   The-name~'#1'~is~already-used-for-a~\token_to_str:N \SubMatrix\
6447   in~this~\@@_full_name_env:.\
6448   If~you-go-on,~this-key~will-be-ignored.\
6449   For-a-list-of-the-names-already-used,~type-H~<return>.
6450 }
6451 {
6452   The-names-already-defined-in-this~\@@_full_name_env:\ are:~
6453   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
6454 }
6455 \@@_msg_new:nn { r-or-l-with-preamble }
6456 {
6457   You-can't-use-the-key~'\l_keys_key_str'~in-your~\@@_full_name_env:~
6458   You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
6459   your~\@@_full_name_env:.\
6460   If~you-go-on,~this-key~will-be-ignored.
6461 }
6462 \@@_msg_new:nn { Hdotsfor~in~col-0 }
6463 {
6464   You-can't-use~\token_to_str:N \Hdotsfor\ in-an-exterior~column-of~
6465   the-array.~This-error-is-fatal.
6466 }
6467 \@@_msg_new:nn { bad-corner }
6468 {
6469   #1-is-an-incorrect-specification-for-a-corner~(in-the-keys~
6470   'corners'~and~'except-corners').~The-available~
6471   values-are:~NW,~SW,~NE~and~SE.\
6472   If~you-go-on,~this-specification-of-corner~will-be-ignored.
6473 }
6474 \@@_msg_new:nn { bad-border }
6475 {
6476   #1-is-an-incorrect-specification-for-a-border~(in-the-key~
6477   'borders'~of-the-command~\token_to_str:N \Block).~The-available~
6478   values-are:~left,~right,~top~and~bottom.\
6479   If~you-go-on,~this-specification-of-border~will-be-ignored.
6480 }
6481 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
6482 {
6483   In-the~\@@_full_name_env:,~you-must-use-the-key~
6484   'last-col'~without-value.\
6485   However,~you-can-go-on~for~this~time~
6486   (the-value~'\l_keys_value_tl'~will-be-ignored).
6487 }
6488 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
6489 {
6490   In~\NiceMatrixoptions,~you-must-use-the-key~
6491   'last-col'~without-value.\
6492   However,~you-can-go-on~for~this~time~
6493   (the-value~'\l_keys_value_tl'~will-be-ignored).
6494 }
6495 \@@_msg_new:nn { Block-too-large-1 }
6496 {
6497   You-try-to-draw-a-block-in-the-cell~#1~#2~of~your~matrix~but~the~matrix~is~
6498   too-small~for~that~block. \
6499 }
6500 \@@_msg_new:nn { Block-too-large-2 }
6501 {
6502   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
6503   \g_@@_static_num_of_col_int\

```

```

6504 columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
6505 specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
6506 (&)~at~the~end~of~the~first~row~of~your~
6507 \@@_full_name_env:.\
6508 If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
6509 }

6510 \@@_msg_new:nn { unknown~column~type }
6511 {
6512   The~column~type~'#1'~in~your~\@@_full_name_env:\
6513   is~unknown. \
6514   This~error~is~fatal.
6515 }

6516 \@@_msg_new:nn { tabularnote~forbidden }
6517 {
6518   You~can't~use~the~command~\token_to_str:N\tabularnote\
6519   ~in~a~\@@_full_name_env:~.~This~command~is~available~only~in~
6520   \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \
6521   If~you~go~on,~this~command~will~be~ignored.
6522 }

6523 \@@_msg_new:nn { borders~forbidden }
6524 {
6525   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
6526   because~the~option~'rounded~corners'~
6527   is~in~force~with~a~non~zero~value.\
6528   If~you~go~on,~this~key~will~be~ignored.
6529 }

6530 \@@_msg_new:nn { bottomrule~without~booktabs }
6531 {
6532   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
6533   loaded~'booktabs'.\
6534   If~you~go~on,~this~key~will~be~ignored.
6535 }

6536 \@@_msg_new:nn { enumitem~not~loaded }
6537 {
6538   You~can't~use~the~command~\token_to_str:N\tabularnote\
6539   ~because~you~haven't~loaded~'enumitem'.\
6540   If~you~go~on,~this~command~will~be~ignored.
6541 }

6542 \@@_msg_new:nn { Wrong~last~row }
6543 {
6544   You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
6545   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
6546   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
6547   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
6548   without~value~(more~compilations~might~be~necessary).
6549 }

6550 \@@_msg_new:nn { Yet~in~env }
6551 { Environments~of~nicematrix~can't~be~nested.\ This~error~is~fatal. }

6552 \@@_msg_new:nn { Outside~math~mode }
6553 {
6554   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
6555   (and~not~in~\token_to_str:N \vcenter).\
6556   This~error~is~fatal.
6557 }

6558 \@@_msg_new:nn { One~letter~allowed }
6559 {
6560   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
6561   If~you~go~on,~it~will~be~ignored.
6562 }

```

```

6563 \@@_msg_new:nnn { Unknown~key~for~Block }
6564 {
6565   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
6566   \Block.\ \ If~you~go~on,~it~will~be~ignored. \ \
6567   For~a~list~of~the~available~keys,~type~H~<return>.
6568 }
6569 {
6570   The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
6571   hvlines,~l,~line-width,~rounded-corners,~r~and~t.
6572 }
6573 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
6574 {
6575   The~key~'\l_keys_key_str'~is~unknown.\ \
6576   If~you~go~on,~it~will~be~ignored. \ \
6577   For~a~list~of~the~available~keys~in~\token_to_str:N
6578   \CodeAfter,~type~H~<return>.
6579 }
6580 {
6581   The~available~keys~are~(in~alphabetic~order):~
6582   delimiters/color,~
6583   rules~(with~the~subkeys~'color'~and~'width'),~
6584   sub-matrix~(several~subkeys)~
6585   and~xdots~(several~subkeys).~
6586   The~latter~is~for~the~command~\token_to_str:N \line.
6587 }
6588 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
6589 {
6590   The~key~'\l_keys_key_str'~is~unknown.\ \
6591   If~you~go~on,~this~key~will~be~ignored. \ \
6592   For~a~list~of~the~available~keys~in~\token_to_str:N
6593   \SubMatrix,~type~H~<return>.
6594 }
6595 {
6596   The~available~keys~are~(in~alphabetic~order):~
6597   'delimiters/color',~
6598   'extra-height',~
6599   'hlines',~
6600   'hvlines',~
6601   'left-xshift',~
6602   'name',~
6603   'right-xshift',~
6604   'rules'~(with~the~subkeys~'color'~and~'width'),~
6605   'slim',~
6606   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
6607   and~'right-xshift').\ \
6608 }
6609 \@@_msg_new:nnn { Unknown~key~for~notes }
6610 {
6611   The~key~'\l_keys_key_str'~is~unknown.\ \
6612   If~you~go~on,~it~will~be~ignored. \ \
6613   For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
6614 }
6615 {
6616   The~available~keys~are~(in~alphabetic~order):~
6617   bottomrule,~
6618   code-after,~
6619   code-before,~
6620   enumitem-keys,~
6621   enumitem-keys-para,~
6622   para,~
6623   label-in-list,~
6624   label-in-tabular~and~
6625   style.

```

```

6626 }
6627 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
6628 {
6629   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
6630   \token_to_str:N \NiceMatrixOptions. \\
6631   If~you~go~on,~it~will~be~ignored. \\
6632   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6633 }
6634 {
6635   The~available~keys~are~(in~alphabetic~order):~
6636   allow~duplicate~names,~
6637   cell-space-bottom-limit,~
6638   cell-space-limits,~
6639   cell-space-top-limit,~
6640   code-for-first-col,~
6641   code-for-first-row,~
6642   code-for-last-col,~
6643   code-for-last-row,~
6644   corners,~
6645   create-extra-nodes,~
6646   create-medium-nodes,~
6647   create-large-nodes,~
6648   delimiters~(several~subkeys),~
6649   end-of-row,~
6650   first-col,~
6651   first-row,~
6652   hlines,~
6653   hvlines,~
6654   last-col,~
6655   last-row,~
6656   left-margin,~
6657   letter-for-dotted-lines,~
6658   light-syntax,~
6659   notes~(several~subkeys),~
6660   nullify-dots,~
6661   renew-dots,~
6662   renew-matrix,~
6663   right-margin,~
6664   rules~(with~the~subkeys~'color'~and~'width'),~
6665   small,~
6666   sub-matrix~(several~subkeys),~
6667   vlines,~
6668   xdots~(several~subkeys).
6669 }
6670 \@@_msg_new:nnn { Unknown~option~for~NiceArray }
6671 {
6672   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
6673   \{NiceArray\}. \\
6674   If~you~go~on,~it~will~be~ignored. \\
6675   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6676 }
6677 {
6678   The~available~keys~are~(in~alphabetic~order):~
6679   b,~
6680   baseline,~
6681   c,~
6682   cell-space-bottom-limit,~
6683   cell-space-limits,~
6684   cell-space-top-limit,~
6685   code-after,~
6686   code-for-first-col,~
6687   code-for-first-row,~
6688   code-for-last-col,~

```

```

6689 code-for-last-row,~
6690 colortbl-like,~
6691 columns-width,~
6692 corners,~
6693 create-extra-nodes,~
6694 create-medium-nodes,~
6695 create-large-nodes,~
6696 delimiters/color,~
6697 extra-left-margin,~
6698 extra-right-margin,~
6699 first-col,~
6700 first-row,~
6701 hlines,~
6702 hvlines,~
6703 last-col,~
6704 last-row,~
6705 left-margin,~
6706 light-syntax,~
6707 name,~
6708 notes/bottomrule,~
6709 notes/para,~
6710 nullify-dots,~
6711 renew-dots,~
6712 right-margin,~
6713 rules~(with~the~subkeys~'color'~and~'width'),~
6714 small,~
6715 t,~
6716 tabularnote,~
6717 vlines,~
6718 xdots/color,~
6719 xdots/shorten~and~
6720 xdots/line-style.
6721 }

```

This error message is used for the set of keys NiceMatrix/NiceMatrix and NiceMatrix/pNiceArray (but not by NiceMatrix/NiceArray because, for this set of keys, there is also the keys t, c and b).

```

6722 \@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
6723 {
6724   The~key~'\l_keys_key_str'~is~unknown~for~the~
6725   \@@_full_name_env:. \\\
6726   If~you~go~on,~it~will~be~ignored. \\\
6727   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6728 }
6729 {
6730   The~available~keys~are~(in~alphabetic~order):~
6731   b,~
6732   baseline,~
6733   c,~
6734   cell-space-bottom-limit,~
6735   cell-space-limits,~
6736   cell-space-top-limit,~
6737   code-after,~
6738   code-for-first-col,~
6739   code-for-first-row,~
6740   code-for-last-col,~
6741   code-for-last-row,~
6742   colortbl-like,~
6743   columns-width,~
6744   corners,~
6745   create-extra-nodes,~
6746   create-medium-nodes,~
6747   create-large-nodes,~
6748   delimiters~(several~subkeys),~

```



```

6749     extra-left-margin,~
6750     extra-right-margin,~
6751     first-col,~
6752     first-row,~
6753     hlines,~
6754     hvlines,~
6755     l,~
6756     last-col,~
6757     last-row,~
6758     left-margin,~
6759     light-syntax,~
6760     name,~
6761     nullify-dots,~
6762     r,~
6763     renew-dots,~
6764     right-margin,~
6765     rules~(with~the~subkeys~'color'~and~'width'),~
6766     small,~
6767     t,~
6768     vlines,~
6769     xdots/color,~
6770     xdots/shorten~and~
6771     xdots/line-style.
6772 }

6773 \@@_msg_new:nnn { Unknown~option~for~NiceTabular }
6774 {
6775     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
6776     \{NiceTabular\}.  \\\
6777     If~you~go~on,~it~will~be~ignored.  \\\
6778     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6779 }
6780 {
6781     The~available~keys~are~(in~alphabetic~order):~
6782     b,~
6783     baseline,~
6784     c,~
6785     cell-space-bottom-limit,~
6786     cell-space-limits,~
6787     cell-space-top-limit,~
6788     code-after,~
6789     code-for-first-col,~
6790     code-for-first-row,~
6791     code-for-last-col,~
6792     code-for-last-row,~
6793     colortbl-like,~
6794     columns-width,~
6795     corners,~
6796     create-extra-nodes,~
6797     create-medium-nodes,~
6798     create-large-nodes,~
6799     extra-left-margin,~
6800     extra-right-margin,~
6801     first-col,~
6802     first-row,~
6803     hlines,~
6804     hvlines,~
6805     last-col,~
6806     last-row,~
6807     left-margin,~
6808     light-syntax,~
6809     name,~
6810     notes/bottomrule,~
6811     notes/para,~

```

```

6812 nullify-dots,~
6813 renew-dots,~
6814 right-margin,~
6815 rules~(with~the~subkeys~'color'~and~'width'),~
6816 t,~
6817 tabularnote,~
6818 vlines,~
6819 xdots/color,~
6820 xdots/shorten~and~
6821 xdots/line-style.
6822 }

6823 \@@_msg_new:nnn { Duplicate-name }
6824 {
6825   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
6826   the~same~environment~name~twice.~You~can~go~on,~but,~
6827   maybe,~you~will~have~incorrect~results~especially~
6828   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
6829   message~again,~use~the~key~'allow-duplicate-names'~in~
6830   '\token_to_str:N \NiceMatrixOptions'.\\
6831   For~a~list~of~the~names~already~used,~type~H~<return>. \\
6832 }
6833 {
6834   The~names~already~defined~in~this~document~are:~
6835   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
6836 }

6837 \@@_msg_new:nn { Option-auto-for-columns-width }
6838 {
6839   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
6840   If~you~go~on,~the~key~will~be~ignored.
6841 }

```

18 History

The successive versions of the file `nicematrix.sty` provided by TeXLive are available on the SVN server of TeXLive:

<https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty>

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types L, C and R.

Changes between version 1.2 and 1.3

New environment `{pNiceArrayC}` and its variants.

Correction of a bug in the definition of `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` (in fact, it was a typo).

Options are now available locally in `{pNiceMatrix}` and its variants.

The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types `w` and `W` can now be used in the environments `{NiceArray}`, `{pNiceArrayC}` and its variants with the same meaning as in the package `array`.

New option `columns-width` to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of `nicematrix` were focused on the continuous dotted lines whereas the version 2.0 of `nicematrix` provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment `{pNiceArrayRC}`. With this new implementation, there is no restriction on the width of the columns.

The package `nicematrix` no longer loads `mathtools` but only `amsmath`.

Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option `code-for-first-row` is now available in the command `\NiceMatrixOptions`.

Following a discussion on TeX StackExchange⁶², Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁶³

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

⁶²cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁶³Before this version, there was an error when using `nicematrix` with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by `nicematrix` because they use the options `overlay` and `remember picture`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`).

Possibility to draw vertical dotted lines to separate columns with the specifier “:” in the preamble (similar to the classical specifier “|” and the specifier “:” of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “:” in the preamble.

Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.

Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of “|”) as `\hdotsfor` does.

Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.

Error message when the user gives an incorrect value for `last-row`.

A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol “:” (in the preamble of the array) and `\line` in `code-after`).

The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.

The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by `colortbl`.

Correction of a bug: it was not possible to use the colon “:” in the preamble of an array when `pdf \LaTeX` 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 StackExchange⁶⁴, 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.

Changes between versions 3.10 and 3.11

Correction of a bug linked to `first-row` and `last-row`.

⁶⁴cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -block and, if the creation of the “medium nodes” is required, a node $i-j$ -block-medium is created.

If the user tries to use more columns than allowed by its environment, an error is raised by `nicematrix` (instead of a low-level error).

The package must be loaded with the option `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the “last row”.

The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customisation of the dotted lines.

In the environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, etc.), it’s possible to use the options `l` (=L) or `r` (=R) to specify the type of the columns.

The starred versions of the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots` and `\Iddots` are deprecated since the version 3.1 of `nicematrix`. Now, one should load `nicematrix` with the option `starred-commands` to avoid an error at the compilation.

The code of `nicematrix` no longer uses Tikz but only PGF. By default, Tikz is *not* loaded by `nicematrix`.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on [stackoverflow](#)).

Better error messages when the user uses `&` or `\\` when `light-syntax` is in force.

Changes between versions 3.14 and 3.15

It’s possible to put labels on the dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and the command `\line` in the `code-after` with the tokens `_` and `^`.

The option `baseline` is now available in all the environments of `nicematrix`. Before, it was available only in `{NiceArray}`.

New keyword `\CodeAfter` (in the environments of `nicematrix`).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`

Commands to color cells, rows and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`

New command `\diagbox`

The key `hvline` don’t draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It’s now possible to write `\begin{pNiceMatrix}a&b\\c&d\end{pNiceMatrix}`² with the expected result.

Changes between versions 4.2 and 4.3

The horizontal centering of the content of a `\Block` is correct even when an instruction such as `!\qqquad` is used in the preamble of the array.

It's now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.

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

Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).

Environment `{NiceTabular*}`

Command `\Vdotsfor` similar to `\Hdotsfor`

The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.

Key `respect-blocks` for `\rowcolors` (with a *s*) in the `code-before`.

The variable `\g_nicematrix_code_before_tl` is now public.

The key `baseline` may take in as value an expression of the form *line-i* to align the `\hline` in the row *i*.

The key `hvlines-except-corners` may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys `c`, `r` and `l` for the command `\Block`.

It's possible to use the key `draw-first` with `\Ddots` and `\Iddots` to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key `tabularnote`.

Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put `\omit` before `\CodeAfter`.

Correction of a bug: the tabular notes `\tabularnotes` were not composed when present in a block (except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.

New command `\NotEmpty`.

Changes between versions 5.6 and 5.7

New key `delimiters-color`

Keys `fill`, `draw` and `line-width` for the command `\Block`.

Changes between versions 5.7 and 5.8

Keys `cols` and `restart` of the command `\rowcolors` in the `code-before`.

Modification of the behaviour of `\\` in the columns of type `p`, `m` or `b` (for a behaviour similar to the environments of `array`).

Better error messages for the command `\Block`.

Changes between versions 5.8 and 5.9

Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.

New key `cell-space-limits`.

Changes between versions 5.9 and 5.10

New command `\SubMatrix` available in the `\CodeAfter`.

It's possible to provide options (between brackets) to the keyword `\CodeAfter`.

A (non fatal) error is raised when the key `transparent`, which is deprecated, is used.

Changes between versions 5.10 and 5.11

It's now possible, in the `code-before` and in the `\CodeAfter`, to use the syntax `|(i-|j)` for the Tikz node at the intersection of the (potential) horizontal rule number i and the (potential) vertical rule number j .

Changes between versions 5.11 and 5.12

Keywords `\CodeBefore` and `\Body` (alternative syntax to the key `code-before`).

New key `delimiters/max-width`.

New keys `hlines`, `vlines` and `hvlines` for the command `\SubMatrix` in the `\CodeAfter`.

New key `rounded-corners` for the command `\Block`.

Changes between versions 5.12 and 5.13

New command `\arraycolor` in the `\CodeBefore` (with its key `except-corners`).

New key `borders` for the command `\Block`.

New command `\Hline` (for horizontal rules not drawn in the blocks).

The keys `vlines` and `hlines` takes in as value a (comma-separated) list of numbers (for the rules to draw).

Changes between versions 5.13 and 5.14

Nodes of the form (1.5) , (2.5) , (3.5) , etc.

Keys `t` and `b` for the command `\Block`.

Key `corners`.

Changes between versions 5.14 and 5.15

Key hvlines for the command \Block.

The commands provided by nicematrix to color cells, rows and columns don't color the cells which are in the "corners" (when the key corner is used).

It's now possible to specify delimiters for submatrices in the preamble of an environment.

Changes between versions 5.15 and 5.16

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

	Symbols	
@@ commands:		
\@@_Block:	1188, 4975	
\@@_Block_i	4980, 4981, 4985	
\@@_Block_ii:nnnnn	4985, 4986	
\@@_Block_iv:nnnnn	5016, 5020	
\@@_Block_iv:nnnnnn	5200, 5202	
\@@_Block_v:nnnnn	5017, 5121	
\@@_Block_v:nnnnnn	5229, 5232	
\@@_Cdots	1113, 1178, 3494	
\g_@@_Cdots_lines_tl	1205, 2647	
\@@_Cell:	201, 836, 1701, 1748, 1770, 2426, 3594, 3595, 3596, 3597, 3598, 3599, 3600, 3601, 3602, 3603, 3604, 3605	
\@@_CodeAfter:	1192, 5753	
\@@_CodeAfter_i:n	838, 2304, 2349, 5753, 5754, 5764	
\@@_CodeAfter_ii:n	5757, 5759	
\@@_CodeAfter_keys:	2585, 2608	
\@@_CodeBefore:w	1380, 1382	
\@@_CodeBefore_keys:	1360, 1377	
\@@_Ddots	1115, 1180, 3526	
\g_@@_Ddots_lines_tl	1208, 2645	
\g_@@_HVDotsfor_lines_tl	1210, 2643, 3653, 3729, 6288	
\@@_Hdotsfor:	1118, 1185, 3629	
\@@_Hdotsfor:nnnn	3655, 3667	
\@@_Hdotsfor_i	3638, 3644, 3651	
\@@_Hline:	1183, 4472	
\@@_Hline_i:n	4472, 4473, 4479	
\@@_Hline_ii:nn	4476, 4479	
\@@_Hline_iii:n	4477, 4480	
\@@_Hspace:	1184, 3580	
\@@_Iddots	1116, 1181, 3550	
\g_@@_Iddots_lines_tl	1209, 2646	
\@@_Ldots	1112, 1117, 1177, 3478	
\g_@@_Ldots_lines_tl	1206, 2648	
\l_@@_Matrix_bool	249, 1560, 1576, 2417	
\l_@@_NiceArray_bool	246, 369, 1270, 1458, 1500, 1613, 1619, 1631, 2393, 4341, 4342, 4464, 4465, 4682, 4689, 5671	
\g_@@_NiceMatrixBlock_int	235, 4741, 4746, 4749, 4760	
\l_@@_NiceTabular_bool	156, 247, 843, 1039, 1359, 1362, 1433, 1533, 1537, 1620, 1632, 2446, 2455, 5048, 5130, 5679	
\@@_NotEmpty:	1194, 2440	
\@@_OnlyMainNiceMatrix:n	1190, 4202	
\@@_OnlyMainNiceMatrix_i:n	4205, 4212, 4215	
\@@_SubMatrix	2567, 5896	
\@@_SubMatrix_in_code_before	1346, 5883	
\@@_Vdots	1114, 1179, 3510	
\g_@@_Vdots_lines_tl	1207, 2644	
\@@_Vdotsfor:	1186, 3727	
\@@_Vdotsfor:nnnn	3731, 3742	
\@@_W:	1579, 1659	
\@@_actually_color:	1361, 3867	
\@@_actually_diagbox:nnnnnn	5027, 5286, 5691, 5707	
\@@_actually_draw_Cdots:	3050, 3054	
\@@_actually_draw_Ddots:	3200, 3204	
\@@_actually_draw_Iddots:	3252, 3256	
\@@_actually_draw_Ldots:	3011, 3015, 3718	
\@@_actually_draw_Vdots:	3132, 3136, 3793	
\@@_adapt_S_column:	171, 186, 1432	
\@@_add_to_colors_seq:nn	3854, 3866, 3889, 3904, 3919, 3928	
\@@_adjust_pos_of_blocks_seq:	2554, 2610	
\@@_adjust_pos_of_blocks_seq_i:nnnn	2613, 2615	
\@@_adjust_size_box:	914, 940, 1779, 2328, 2373	
\@@_adjust_to_submatrix:nn	2895, 2998, 3037, 3118, 3193, 3245	
\@@_adjust_to_submatrix:nnnnnn	2902, 2904	
\@@_after_array:	1569, 2459	
\g_@@_after_col_zero_bool	275, 1080, 2305, 3635	
\@@_analyze_end:Nn	2102, 2147	
\l_@@_argspec_tl	3476, 3477, 3478, 3494, 3510, 3526, 3550, 3649, 3650, 3651, 3725, 3726, 3727, 3803, 3804, 3805	
\@@_array:	1034, 2103, 2130	
\@@_arraycolor	1343, 3965	
\c_@@_arydshln_loaded_bool	24, 31, 1687	
\l_@@_auto_columns_width_bool	472, 612, 2214, 2218, 4736	

<code>\l_@@_baseline_tl</code>	463, 464, 605, 606, 607, 608, 1047, 1502, 1915, 1927, 1932, 1934, 1939, 1944, 2026, 2027, 2031, 2036, 2038, 2043	<code>\l_@@_colortbl_like_bool</code>	449, 628, 1195, 1602
<code>\@@_begin_of_NiceMatrix:nn</code>	2415, 2436	<code>\c_@@_colortbl_loaded_bool</code>	88, 92, 1145
<code>\@@_begin_of_row:</code>	841, 864, 2306	<code>\l_@@_cols_tl</code>	3897, 3912, 3942, 3976, 3984, 3985, 4090, 4093
<code>\l_@@_block_auto_columns_width_bool</code>	1446, 2219, 4729, 4734, 4744, 4754	<code>\g_@@_cols_vlism_seq</code>	258, 1597, 1678, 5988
<code>\g_@@_block_box_int</code>	313, 1426, 5022, 5036, 5038, 5074, 5086, 5098, 5107, 5117	<code>\@@_columncolor</code>	1344, 3900
<code>\g_@@_blocks_dp_dim</code>	243, 922, 925, 926, 5101, 5104	<code>\@@_columncolor:n</code>	3906, 3909
<code>\g_@@_blocks_ht_dim</code>	242, 928, 931, 932, 5092, 5095	<code>\@@_columncolor_preamble</code>	1124, 4190
<code>\g_@@_blocks_seq</code>	289, 1448, 1965, 5111, 5123, 5200	<code>\c_@@_columncolor_regex</code>	210, 1605
<code>\g_@@_blocks_wd_dim</code>	241, 916, 919, 920, 5080, 5083	<code>\l_@@_columns_width_dim</code>	236, 613, 755, 2215, 2221, 4742, 4748
<code>\c_@@_booktabs_loaded_bool</code>	25, 34, 1128, 1997	<code>\g_@@_com_or_env_str</code>	262, 265
<code>\l_@@_borders_clist</code>	303, 5179, 5259, 5547, 5563, 5565, 5567, 5569, 5588, 5600	<code>\@@_computations_for_large_nodes:</code>	4858, 4871, 4876
<code>\@@_cartesian_path:</code>	3898, 3913, 4037, 4049, 4142	<code>\@@_computations_for_medium_nodes:</code>	4778, 4847, 4857, 4868
<code>\@@_cartesian_path:n</code>	3943, 4084, 4142	<code>\@@_compute_a_corner:nnnnnn</code>	4543, 4545, 4547, 4549, 4565
<code>\l_@@_cell_box</code>	842, 888, 890, 896, 902, 905, 909, 918, 919, 924, 925, 930, 931, 941, 942, 943, 944, 946, 949, 953, 955, 974, 989, 991, 998, 999, 1012, 1130, 1222, 1224, 1769, 1780, 2307, 2331, 2334, 2336, 2353, 2376, 2380, 5294, 5398, 5432, 5686	<code>\@@_compute_corners:</code>	2553, 4535
<code>\l_@@_cell_space_bottom_limit_dim</code>	452, 524, 944	<code>\@@_construct_preamble:</code>	1267, 1573
<code>\l_@@_cell_space_top_limit_dim</code>	451, 522, 942	<code>\l_@@_corners_cells_seq</code>	293, 1350, 2559, 4087, 4127, 4276, 4282, 4289, 4401, 4407, 4414, 4537, 4553, 4560, 4621
<code>\l_@@_cell_type_tl</code>	239, 240, 1701, 1771, 5000, 5002	<code>\l_@@_corners_clist</code>	468, 590, 595, 4242, 4368, 4538
<code>\@@_cellcolor</code>	1338, 3945, 3957, 3958	<code>\@@_create_col_nodes:</code>	2106, 2134, 2153
<code>\@@_cellcolor_tabular</code>	1122, 4167	<code>\@@_create_diag_nodes:</code>	1327, 2530, 2714
<code>\g_@@_cells_seq</code>	2141, 2142, 2143, 2145	<code>\@@_create_extra_nodes:</code>	1964, 2706, 4768
<code>\@@_chessboardcolors</code>	1345, 3950	<code>\@@_create_large_nodes:</code>	4776, 4852
<code>\@@_cline</code>	135, 1176	<code>\@@_create_medium_and_large_nodes:</code>	4773, 4863
<code>\@@_cline_i:nn</code>	136, 137, 149, 152	<code>\@@_create_medium_nodes:</code>	4774, 4842
<code>\@@_cline_i:w</code>	137, 138	<code>\@@_create_nodes:</code>	4849, 4860, 4870, 4873, 4914
<code>\l_@@_code_before_bool</code>	279, 602, 629, 1054, 1216, 1292, 1454, 2161, 2178, 2196, 2227, 2253, 2280, 2505, 2601, 2603	<code>\@@_create_row_node:</code>	1050, 1083, 1129
<code>\l_@@_code_before_tl</code>	278, 601, 1291, 1360, 1455	<code>\@@_cut_on_hyphen:w</code>	3880, 3935, 3940, 4005, 4097, 4098, 4120, 4121, 4151, 4152, 5465, 5474, 5505, 5508, 5552, 5555, 5885, 5890, 5903, 5906
<code>\l_@@_code_for_first_col_tl</code>	541, 2318	<code>\g_@@_ddots_int</code>	2533, 3224, 3225
<code>\l_@@_code_for_first_row_tl</code>	545, 852, 5370	<code>\@@_def_env:nnn</code>	2399, 2410, 2411, 2412, 2413, 2414
<code>\l_@@_code_for_last_col_tl</code>	543, 2362	<code>\@@_define_L_C_R:</code>	223, 1266
<code>\l_@@_code_for_last_row_tl</code>	547, 859, 5373	<code>\c_@@_define_L_C_R_bool</code>	222, 1266, 6201
<code>\l_@@_code_tl</code>	270, 5878, 6085	<code>\@@_define_com:nnn</code>	5655, 5663, 5664, 5665, 5666, 5667
<code>\l_@@_col_max_int</code>	298, 2762, 2773, 2841, 2900, 2917	<code>\@@_delimiter:nnn</code>	1806, 1817, 1839, 1847, 1860, 1866, 1872, 5767
<code>\l_@@_col_min_int</code>	297, 2767, 2830, 2835, 2898, 2915	<code>\l_@@_delimiters_color_tl</code>	484, 688, 1373, 1525, 1526, 1543, 1544, 5747, 5805, 5806, 5833, 6137, 6138, 6161, 6162
<code>\g_@@_col_total_int</code>	238, 957, 1201, 1486, 2246, 2247, 2283, 2287, 2292, 2293, 2352, 2463, 2466, 2471, 2478, 2522, 2672, 3085, 3103, 3163, 3625, 3626, 3788, 4192, 4787, 4797, 4831, 4918, 5212, 5405, 5911, 5960	<code>\l_@@_delimiters_max_width_bool</code>	485, 686, 1548
<code>\l_@@_color_tl</code>	305, 4972, 5040, 5042	<code>\g_@@_delta_x_one_dim</code>	2535, 3227, 3237
<code>\g_@@_colors_seq</code>	1356, 3857, 3861, 3862, 3871	<code>\g_@@_delta_x_two_dim</code>	2537, 3275, 3285
<code>\@@_colortbl_like:</code>	1120, 1195	<code>\g_@@_delta_y_one_dim</code>	2536, 3229, 3237
		<code>\g_@@_delta_y_two_dim</code>	2538, 3277, 3285
		<code>\@@_diagbox:nn</code>	1193, 5687
		<code>\@@_dotfill:</code>	5676
		<code>\@@_dotfill_i:</code>	5681, 5683
		<code>\@@_dotfill_ii:</code>	5680, 5683, 5684
		<code>\@@_dotfill_iii:</code>	5684, 5685
		<code>\@@_double_int_eval:n</code>	3799, 3813, 3814
		<code>\g_@@_dp_ante_last_row_dim</code>	867, 1161

\g_@@_dp_last_row_dim	833, 834, 1375, 1481, 1491, 1563, 1949,
.... 867, 868, 1164, 1165, 1223, 1224, 1519	2002, 2048, 3964, 3979, 5195, 5545, 5751,
\g_@@_dp_row_zero_dim	5863, 5874, 5881, 5912, 6211, 6216, 6243, 6253
.... 887, 888, 1155, 1156, 1512, 2020, 2059	\@@_error:nn
\@@_draw_Cdots:nnn 13, 620, 1667, 1668, 1669, 1809,
\@@_draw_Ddots:nnn	1820, 1867, 1873, 3481, 3484, 3497, 3500,
\@@_draw_Iddots:nnn	3513, 3516, 3530, 3531, 3536, 3537, 3554,
\@@_draw_Ldots:nnn	3555, 3560, 3561, 4551, 5550, 5868, 5942, 5945
\@@_draw_Vdots:nnn	\@@_error:nnn
\@@_draw_blocks: 14, 3830, 6017, 6052
\@@_draw_dotted_lines:	\@@_error_too_much_cols:
\@@_draw_dotted_lines_i: 1641, 6273
\l_@@_draw_first_bool . 311, 3541, 3565, 3576	\@@_everycr:
\@@_draw_hlines: 1076, 1150, 1153
\@@_draw_line:	\@@_everycr_i:
3078, 3189, 3241, 3289, 3291, 3852, 4697, 4727 1076, 1077
\@@_draw_line_ii:nn	\@@_exec_code_before:
\@@_draw_line_iii:nn 1216, 1354
\@@_draw_non_standard_dotted_line: ..	\@@_expand_clist:NN
..... 3297, 3299 4090, 4091, 4143
\@@_draw_non_standard_dotted_line:n ..	\l_@@_exterior_arraycolsep_bool
..... 3302, 3305 465, 751, 1622, 1634
\@@_draw_non_standard_dotted_line:nnn	\l_@@_extra_left_margin_dim
..... 3307, 3312, 3326 479, 579, 1283, 2339
\@@_draw_standard_dotted_line: . 3296, 3327	\l_@@_extra_right_margin_dim
\@@_draw_standard_dotted_line_i: 3390, 3394 480, 580, 1473, 2384, 3166
\l_@@_draw_tl	\@@_extract_brackets
302, 5173, 5271, 5437
5177, 5237, 5447, 5453, 5455, 5457, 5494, 5495	\@@_extract_coords_values:
\@@_draw_vlines: 4939, 4946
2566, 4338	\@@_fatal:n
\g_@@_empty_cell_bool 15, 254, 1436, 1794, 1828,
286, 948, 958,	2111, 2115, 2117, 2150, 3640, 6278, 6281, 6284
2341, 2388, 3492, 3508, 3524, 3548, 3571, 3582	\@@_fatal:nn
\@@_end_Cell: 16, 1692
203, 935, 1703,	\l_@@_fill_tl
1758, 1775, 2426, 3594, 3595, 3596, 3597, 301, 5171, 5269, 5271
3598, 3599, 3600, 3601, 3602, 3603, 3604, 3605	\l_@@_final_i_int
\l_@@_end_of_row_tl 2542, 2749, 2754, 2757, 2782,
..... 481, 482, 535, 2126, 2127, 6386	2790, 2794, 2803, 2811, 2891, 2946, 3027,
\c_@@_endpgfortikzpicture_tl	3100, 3106, 3109, 3672, 3700, 3768, 3778, 3780
..... 43, 47, 2636, 3840, 4668	\l_@@_final_j_int
\c_@@_enumitem_loaded_bool 2543, 2750, 2755, 2762, 2767, 2773, 2783,
..... 26, 37, 342, 656, 661, 672, 677	2791, 2795, 2804, 2812, 2892, 2947, 2980,
\@@_env:	2983, 2991, 3217, 3693, 3703, 3705, 3747, 3776
..... 230, 234,	\l_@@_final_open_bool
873, 879, 975, 981, 995, 1003, 1059, 1065, 2545, 2756,
1071, 1307, 1308, 1315, 1316, 1323, 1324,	2760, 2763, 2770, 2776, 2780, 2796, 3024,
1335, 1401, 1404, 1407, 1410, 2162, 2165,	3059, 3064, 3075, 3139, 3149, 3154, 3175,
2167, 2183, 2189, 2192, 2201, 2207, 2210,	3214, 3264, 3398, 3413, 3444, 3445, 3670,
2232, 2238, 2241, 2258, 2264, 2270, 2283,	3694, 3706, 3745, 3769, 3781, 3822, 4665, 4702
2287, 2293, 2576, 2674, 2678, 2684, 2691,	\@@_find_extremities_of_line:nnnn ...
2697, 2731, 2738, 2740, 2741, 2802, 2870, 2744, 3001, 3040, 3121, 3196, 3248
2934, 2945, 2958, 2961, 2980, 2983, 3088,	\l_@@_first_col_int
3091, 3106, 3109, 3681, 3699, 3756, 3774, 123, 136, 323, 324, 537,
3825, 3827, 3846, 3849, 4576, 4595, 4613,	810, 841, 1495, 1614, 2156, 2176, 2516,
4800, 4802, 4810, 4921, 4930, 4948, 5309,	2672, 3085, 3103, 3633, 4111, 4204, 4787,
5316, 5320, 5334, 5339, 5351, 5356, 5357,	4797, 4831, 4879, 4918, 5637, 5643, 5649, 5960
5360, 5377, 5411, 5782, 5785, 5927, 5929,	\l_@@_first_i_tl
5934, 5936, 5963, 5965, 5970, 5972, 6070, 6081 5899, 5904, 5923, 5954,
\g_@@_env_int ..	5963, 5965, 6020, 6027, 6029, 6110, 6121, 6125
229, 230, 232, 1232, 1235,	\l_@@_first_j_tl
1250, 1253, 1299, 1302, 1312, 1313, 1320, 5900, 5905, 5927,
1321, 1348, 1351, 1384, 1393, 1394, 1397,	5929, 5990, 6003, 6010, 6012, 6111, 6122, 6126
1398, 1445, 1452, 1456, 2470, 2491, 2509,	\l_@@_first_row_int
2512, 2525, 2597, 3992, 3995, 4020, 4559, 4957 321, 322, 538,
\@@_error:n	814, 1199, 1510, 1946, 2017, 2045, 2056,
..... 12, 345, 370, 494, 504,	2514, 2670, 2955, 2977, 4780, 4794, 4821,
557, 682, 738, 748, 754, 763, 771, 789, 796,	4878, 4916, 5313, 5331, 5635, 5779, 5924, 6404
804, 805, 806, 812, 817, 818, 819, 830, 832,	\c_@@_footnote_bool
 1421, 1571, 6178, 6214, 6237, 6240, 6250, 6256
	\c_@@_footnotehyper_bool . 6177, 6215, 6247
	\@@_full_name_env:
 263, 6262, 6294, 6301, 6309, 6317, 6321,
	6418, 6431, 6434, 6447, 6452, 6457, 6459,
	6483, 6502, 6507, 6512, 6519, 6545, 6554, 6725
	\@@_hdottedline:
 1182, 4650
	\@@_hdottedline:n
 4658, 4662

\@@_hdottedline_i:	4653, 4655	3418, 3426, 3434, 3439, 3451, 3452, 3459, 3460
\@@_hdottedline_i:n	4667, 4671	\l_@@_large_nodes_bool 476, 570, 4772, 4776
\@@_hline:nn	4349, 4469, 4488	\g_@@_last_col_found_bool .. 331, 1204,
\@@_hline_i:nn	2563, 4352, 4355	1487, 1555, 2245, 2274, 2350, 2462, 2519, 5403
\@@_hline_i_complete:nn	2563, 4459	\l_@@_last_col_int
\@@_hline_ii:nnnn	4377, 4388, 4421, 4460	329,
\l_@@_hlines_clist	317, 549,	330, 739, 782, 784, 797, 813, 831, 1246,
563, 592, 1084, 1086, 1090, 2565, 4467, 4468		1252, 1259, 1305, 1490, 1626, 2422, 2424,
\l_@@_hpos_of_block_tl ..	307, 308, 4962,	2463, 2466, 2518, 3126, 3161, 3483, 3499,
4964, 4966, 5001, 5002, 5004, 5047, 5053,		3537, 3561, 5205, 5210, 5211, 5212, 5215,
5063, 5114, 5137, 5141, 5153, 5158, 5185,		5244, 5256, 5266, 5278, 5290, 5305, 5334,
5187, 5189, 5379, 5391, 5402, 5406, 5413, 5425		5339, 5347, 5362, 5639, 5645, 5651, 6277, 6295
\g_@@_ht_last_row_dim		\l_@@_last_col_without_value_bool ..
869, 1162, 1163, 1221, 1222, 1518		328, 781, 2464, 6280
\g_@@_ht_row_one_dim ..	895, 896, 1159, 1160	\l_@@_last_empty_column_int
\g_@@_ht_row_zero_dim		4586, 4587, 4600, 4606, 4619
889, 890, 1157, 1158, 1513, 2019, 2058		\l_@@_last_empty_row_int
\@@_hvlines_block:nnn	5253, 5501	4568, 4569, 4582, 4603
\l_@@_hvlines_block_bool ..	312, 5181, 5249	\l_@@_last_i_tl
\@@_i:	4780, 4782,	5901,
4783, 4784, 4785, 4794, 4800, 4802, 4803,		5907, 5910, 5923, 5957, 5970, 5972, 6020, 6027
4804, 4805, 4810, 4811, 4812, 4813, 4821,		\l_@@_last_j_tl
4824, 4826, 4827, 4828, 4880, 4882, 4885,		5902, 5908, 5911, 5934, 5936, 5993, 6003, 6010
4886, 4890, 4891, 4916, 4921, 4923, 4925,		\l_@@_last_row_int
4929, 4930, 4941, 4948, 4950, 4952, 4956, 4957		325, 326, 539, 857, 903, 1096, 1217, 1227,
\g_@@_iddots_int	2534, 3272, 3273	1234, 1241, 1303, 1475, 1479, 1482, 1494,
\l_@@_in_env_bool	245, 369, 1436, 1437	1516, 2128, 2129, 2314, 2315, 2359, 2360,
\c_@@_in_preamble_bool .	21, 22, 23, 652, 668	2485, 3006, 3045, 3515, 3531, 3555, 4210,
\l_@@_initial_i_int	2540,	4218, 5204, 5207, 5208, 5227, 5244, 5256,
2747, 2822, 2825, 2850, 2858, 2862, 2871,		5266, 5278, 5289, 5303, 5361, 5372, 5647, 6544
2879, 2889, 2935, 3020, 3066, 3068, 3082,		\l_@@_last_row_without_value_bool ..
3088, 3091, 3671, 3672, 3682, 3750, 3760, 3762		327, 1229, 1477, 2483
\l_@@_initial_j_int		\l_@@_left_delim_dim
2541, 2748, 2823, 2830,		1268, 1272, 1277, 2094, 2337
2835, 2841, 2851, 2859, 2863, 2872, 2880,		\g_@@_left_delim_tl
2890, 2936, 2958, 2961, 2969, 3156, 3158,		1276, 1423, 1527, 1551, 1611, 1797, 1799, 4692
3163, 3209, 3675, 3685, 3687, 3746, 3747, 3758		\l_@@_left_margin_dim
\l_@@_initial_open_bool		477, 573, 1282, 2338, 4683, 4909
2544, 2824, 2828, 2831, 2838, 2844,		\l_@@_letter_for_dotted_lines_str ..
2848, 2864, 3017, 3056, 3063, 3073, 3139,		762, 773, 774, 1673
3146, 3152, 3206, 3258, 3396, 3443, 3669,		\l_@@_letter_vlism_tl
3676, 3688, 3744, 3751, 3763, 3821, 4664, 4701		257, 556, 1676
\@@_insert_tabularnotes:	1969, 1972	\l_@@_light_syntax_bool
\@@_instruction_of_type:nnn		462, 533, 1285, 1468, 2486
1015, 3486, 3502, 3518, 3541, 3565		\@@_light_syntax_i
\c_@@_integers_alist_tl	6095, 6106	2119, 2122
\l_@@_inter_dots_dim		\@@_line
453, 454, 2549, 3401, 3408, 3419, 3427,		2580, 3805
3434, 3439, 3451, 3459, 4693, 4695, 4723, 4725		\@@_line_i:nn
\g_@@_internal_code_after_tl		3812, 3819
271, 1737, 1805,		\l_@@_line_width_dim
1816, 1838, 1846, 1859, 1865, 1871, 1881,		306, 5183, 5448, 5486, 5497, 5503, 5515,
2581, 2582, 4487, 4657, 5025, 5284, 5689, 5893		5542, 5562, 5577, 5579, 5589, 5590, 5592, 5603
\@@_intersect_our_row:nnnn	4073	\@@_line_with_light_syntax:n ..
\@@_intersect_our_row_p:nnnn	4024	2133, 2137
\@@_j:	4787, 4789,	\@@_line_with_light_syntax_i:n
4790, 4791, 4792, 4797, 4800, 4802, 4805,		2132, 2138, 2139
4807, 4808, 4810, 4813, 4815, 4816, 4831,		\@@_math_toggle_token:
4834, 4836, 4837, 4838, 4893, 4895, 4898,		155, 937, 2308, 2325, 2354, 2370, 5731, 5735
4900, 4904, 4905, 4918, 4921, 4922, 4924,		\g_@@_max_cell_width_dim
4929, 4930, 4942, 4948, 4949, 4951, 4956, 4957		945, 946, 1447, 2220, 4735, 4761
\l_@@_l_dim		\c_@@_max_l_dim
3374, 3375, 3388, 3389, 3401, 3407,		3388, 3393
		\l_@@_medium_nodes_bool 475, 569, 4770, 5353
		\@@_message_hdotsfor: 6286, 6294, 6301, 6309
		\@@_msg_new:nn
		17, 6193, 6219, 6228, 6258, 6291,
		6298, 6306, 6314, 6319, 6325, 6330, 6335,
		6340, 6345, 6350, 6355, 6360, 6366, 6372,
		6377, 6383, 6389, 6394, 6401, 6408, 6415,
		6421, 6430, 6432, 6438, 6455, 6462, 6467,

6474, 6481, 6488, 6495, 6500, 6510, 6516, 6523, 6530, 6536, 6542, 6550, 6552, 6558, 6837	\@@_open_x_initial_dim:
\@@_msg_new:nnn ... 18, 6179, 6444, 6563, 6573, 6588, 6609, 6627, 6670, 6722, 6773, 6823 2952, 3019, 3057, 3211, 3261
\@@_msg_redirect_name:nn	\@@_open_y_final_dim: 3098, 3150, 3216, 3266
..... 19, 757, 1567, 5218, 5221	\@@_open_y_initial_dim:
\@@_multicolumn:nnn 3080, 3147, 3208, 3260
..... 1187, 3586	\l_@@_parallelize_diags_bool
\g_@@_multicolumn_cells_seq 466, 467, 565, 2531, 3222, 3270
... 1197, 3613, 4805, 4813, 4935, 5318, 5336	\@@_patch_node_for_cell:
\g_@@_multicolumn_sizes_seq 1198, 3615, 4936 987, 1365
\g_@@_name_env_str	\@@_patch_node_for_cell:n . 985, 1011, 1014
..... 261, 266, 267, 1430, 1431, 2149, 2394, 2395, 2403, 2404, 2433, 2444, 2452, 2604, 5659, 6275	\@@_patch_preamble:n
\l_@@_name_str .. 474, 622, 875, 878, 977, 980, 1067, 1070, 1230, 1239, 1242, 1248, 1257, 1260, 2166, 2167, 2191, 2192, 2209, 2210, 2240, 2241, 2266, 2269, 2289, 2292, 2473, 2477, 2494, 2498, 4926, 4929, 4953, 4956 1599, 1644, 1682, 1690, 1711, 1740, 1801, 1810, 1812, 1821, 1823, 1840, 1848, 1874, 1883, 1903
\g_@@_names_seq	\@@_patch_preamble_i:n 1648, 1649, 1650, 1697
..... 244, 619, 621, 6835	\@@_patch_preamble_ii:nn
\l_@@_nb_cols_int 1651, 1652, 1653, 1708
..... 5626, 5631, 5634, 5638, 5644, 5650	\@@_patch_preamble_iii:n . 1654, 1713, 1721
\l_@@_nb_rows_int	\@@_patch_preamble_iii_i:n 1716, 1718
..... 5625, 5630, 5641	\@@_patch_preamble_iv:nnn
\@@_newcolumnmtype 1655, 1656, 1657, 1743
..... 1103, 1578, 1579	\@@_patch_preamble_ix:n
\@@_node_for_cell: 954, 961, 1365, 2335, 2385 1888, 1906
\@@_node_for_multicolumn:nn 4937, 4944	\@@_patch_preamble_v:nnnn 1658, 1659, 1764
\@@_node_left:nn	\@@_patch_preamble_vi:n
..... 6069, 6070, 6129 1660, 1786
\@@_node_position:	\@@_patch_preamble_vii:nn
..... 1315, 1317, 1323, 1325, 1405, 1411 1661, 1662, 1663, 1792
\@@_node_position_i:	\@@_patch_preamble_viii:nn
..... 1408, 1412 1664, 1665, 1666, 1826
\@@_node_right:nn	\@@_patch_preamble_viii_i:nnn .. 1830, 1852
..... 6079, 6081, 6153	\@@_patch_preamble_x:n
\g_@@_not_empty_cell_bool 277, 952, 959, 2441 1706, 1762, 1784, 1790, 1885, 1909
\@@_not_in_exterior:nnnn	\@@_patch_preamble_xi:n
..... 4065 1674, 1877
\@@_not_in_exterior_p:nnnn	\@@_pgf_rect_node:nnn 425, 1409, 5355
..... 3997	\@@_pgf_rect_node:nnnnn
\l_@@_notes_above_space_dim 469, 470 400, 4920, 4947, 5308, 5350, 6056
\l_@@_notes_bottomrule_bool	\c_@@_pgfortikzpicture_tl
..... 640, 800, 825, 1995 42, 46, 2634, 3838, 4666
\l_@@_notes_code_after_tl	\@@_pgfpointanchor:n
..... 638, 2004 6083, 6088
\l_@@_notes_code_before_tl 636, 1976	\@@_pgfpointanchor_i:nn
\@@_notes_label_in_list:n 338, 357, 365, 648 6091, 6093
\@@_notes_label_in_tabular:n . 337, 378, 645	\@@_pgfpointanchor_ii:w
\l_@@_notes_para_bool .. 634, 798, 823, 1980 6094, 6102
\@@_notes_style:n	\@@_pgfpointanchor_iii:w
..... 336, 339, 357, 365, 381, 386, 642 6115, 6117
\l_@@_nullify_dots_bool	\@@_picture_position:
..... 471, 568, 3490, 3506, 3522, 3546, 3569 1308, 1317, 1325, 1411, 1412
\l_@@_number_of_notes_int 335, 372, 382, 392	\g_@@_pos_of_blocks_seq 290, 1449, 2526, 2557, 2612, 3616, 4236, 4362, 4633, 5234, 5699
\@@_old_CT@arc@	\g_@@_pos_of_stroken_blocks_seq
..... 1438, 2606 292, 1450, 4240, 4366, 5246
\@@_old_cdots	\g_@@_pos_of_xdots_seq
..... 1170, 3507 291, 1451, 2558, 2887, 4238, 4364
\@@_old_ddots	\@@_pre_array:
..... 1172, 3547 1213, 1293, 1465
\@@_old_dotfill	\@@_pre_array_i:w
..... 5675, 5678, 5686 1289, 1465
\@@_old_dotfill:	\@@_pre_array_ii:
..... 1191 1126, 1264
\l_@@_old_iRow_int	\@@_pre_code_before:
..... 272, 1132, 2652 1295, 1386
\@@_old_ialign:	\c_@@_preamble_first_col_tl 1615, 2300
..... 1049, 1166, 5199	\c_@@_preamble_last_col_tl 1627, 2345
\@@_old_iddots	\g_@@_preamble_tl
..... 1173, 3570 1425, 1580, 1584, 1587, 1593, 1607, 1615, 1624, 1627, 1636, 1640, 1680, 1689, 1699, 1710, 1723, 1745, 1766, 1788, 1804, 1837, 1845, 1858, 1864, 1879, 1892, 1899, 1908, 2103, 2130
\l_@@_old_jCol_int	\@@_pred:n
..... 273, 1135, 2653 124, 154, 2424, 4277, 4290, 4402, 4415
\@@_old_ldots	\@@_provide_pgfsyspdfmark: . 211, 220, 1420
..... 1169, 3491	\@@_put_box_in_flow:
\@@_old_multicolumn 1553, 1911, 2096
..... 3585, 3589	\@@_put_box_in_flow_bis:nn 1550, 2063
\@@_old_pgfpointanchor 163, 6087, 6091	
\@@_old_pgful@check@rerun	
..... 81, 85	
\@@_old_vdots	
..... 1171, 3523	
\@@_open_x_final_dim:	
..... 2974, 3026, 3060, 3218, 3267	

\@@_put_box_in_flow_i:	1917, 1919	\l_@@_rules_color_tl	274, 508, 1462, 1463, 5985, 5986
\@@_qpoint:n	233, 1922, 1924, 1936,	\@@_set_CT@arc@:	157, 1463, 5986
1952, 2011, 2013, 2029, 2040, 2051, 2720,		\@@_set_CT@arc@_i:	158, 159
2722, 2724, 2726, 2734, 2736, 2969, 2991,		\@@_set_CT@arc@_ii:	158, 161
3020, 3027, 3066, 3068, 3082, 3100, 3156,		\@@_set_final_coords:	2925, 2950
3158, 3209, 3217, 3846, 3849, 4110, 4114,		\@@_set_final_coords_from_anchor:n ..	
4130, 4132, 4300, 4302, 4304, 4425, 4427,		... 2941, 3030, 3061, 3142, 3151, 3221, 3269	
4429, 4674, 4678, 4685, 4719, 4722, 4724,		\@@_set_initial_coords:	2920, 2939
4826, 4836, 5299, 5301, 5303, 5305, 5327,		\@@_set_initial_coords_from_anchor:n ..	
5347, 5375, 5470, 5472, 5479, 5481, 5516,		... 2930, 3023, 3058, 3141, 3148, 3213, 3263	
5518, 5522, 5527, 5529, 5533, 5576, 5578,		\@@_set_size:n	5623, 5632
5580, 5587, 5591, 5593, 5712, 5714, 5717,		\c_@@_siunitx_loaded_bool	164, 168, 173, 191
5719, 5772, 5774, 5954, 5957, 5995, 6012, 6029		\l_@@_small_bool	740, 787, 793,
\l_@@_radius_dim	457, 458, 1880,	815, 846, 1138, 1794, 1828, 2309, 2355, 2546	
2548, 3031, 3032, 3468, 4652, 4676, 4720, 4721		\@@_standard_cline	120, 1175
\l_@@_real_left_delim_dim	2065, 2080, 2095	\@@_standard_cline:w	120, 121
\l_@@_real_right_delim_dim	2066, 2092, 2098	\l_@@_standard_cline_bool ..	450, 520, 1174
\@@_recreate_cell_nodes:	1328, 1390	\c_@@_standard_tl	460, 461, 3295, 4696, 4726
\g_@@_recreate_cell_nodes_bool		\g_@@_static_num_of_col_int	
..... 473, 1328, 1357, 1364, 1369		... 300, 1562, 1600, 5215, 6310, 6322, 6503	
\@@_rectanglecolor ..	1339, 3915, 3948, 3968	\l_@@_stop_loop_bool	2751, 2752,
\@@_rectanglecolor:nmn ...	3921, 3930, 3933	2784, 2797, 2806, 2819, 2820, 2852, 2865, 2874	
\@@_renew_NC@rewrite@S:	192, 194, 1203	\@@_store_in_tmpb_tl	5440, 5442
\@@_renew_dots:	1110, 1196	\@@_stroke_block:nmn	5241, 5444
\l_@@_renew_dots_bool		\@@_stroke_borders_block:nmn ...	5263, 5540
..... 566, 747, 1196, 6203, 6210		\@@_stroke_horizontal:n ..	5568, 5570, 5585
\@@_renew_matrix:	742, 746, 5605, 6205, 6209	\@@_stroke_vertical:n	5564, 5566, 5574
\l_@@_respect_blocks_bool	3974, 3989, 4017	\@@_sub_matrix_i:nn	5946, 5952
\@@_restore_iRow_jCol:	2605, 2650	\l_@@_submatrix_extra_height_dim	
\@@_revtex_array:	1026, 1037 314, 5825, 5980	
\c_@@_revtex_bool	50, 52, 55, 57, 1036	\l_@@_submatrix_hlines_clist	
\l_@@_right_delim_dim 319, 5837, 5855, 6019, 6021	
..... 1269, 1273, 1279, 2097, 2382		\l_@@_submatrix_left_xshift_dim	
\g_@@_right_delim_tl ..	1278, 1424, 1545, 315, 5827, 6032, 6065	
1551, 1612, 1800, 1834, 1835, 1856, 1861, 4694		\l_@@_submatrix_name_str	
\l_@@_right_margin_dim		5870, 5914, 6054, 6056, 6068, 6070, 6078, 6081	
... 478, 575, 1472, 2383, 3165, 4690, 4912		\g_@@_submatrix_names_seq	
\@@_rotate:	1189, 3798 294, 2584, 5867, 5871, 6453	
\g_@@_rotate_bool		\l_@@_submatrix_right_xshift_dim	
..... 250, 912, 939, 1778, 2327,	 316, 5829, 6041, 6075	
2372, 3798, 5047, 5071, 5076, 5136, 5152, 5295		\g_@@_submatrix_seq ...	299, 1215, 2901, 5891
\@@_rotate_cell_box:		\l_@@_submatrix_slim_bool	5835, 5922, 6426
..... 900, 939, 1778, 2327, 2372, 5295		\l_@@_submatrix_vlines_clist	
\l_@@_rounded_corners_dim 320, 5839, 5857, 6002, 6004	
304, 5175, 5279, 5462, 5463, 5498, 5544, 5601		\@@_succ:n	149,
\@@_roundedrectanglecolor	1340, 3924	153, 1059, 1065, 1091, 1738, 1806, 1817,	
\l_@@_row_max_int	296, 2757, 2899, 2916	1897, 1924, 2258, 2264, 2269, 2270, 2283,	
\l_@@_row_min_int	295, 2825, 2897, 2914	2287, 2292, 2293, 2520, 2991, 3068, 3158,	
\g_@@_row_of_col_done_bool		3217, 4069, 4114, 4130, 4273, 4304, 4342,	
..... 276, 1081, 1429, 2175		4398, 4429, 4465, 4488, 4638, 4640, 4642,	
\g_@@_row_total_int	237, 1200,	4644, 4685, 4724, 4886, 4890, 4900, 4904,	
1493, 1947, 2046, 2485, 2492, 2499, 2515,		5303, 5305, 5347, 5479, 5481, 5717, 5719, 5774	
2670, 2955, 2977, 3713, 4780, 4794, 4821,		\l_@@_suffix_tl	4848, 4859,
4916, 5227, 5313, 5331, 5779, 5910, 5924, 6405		4869, 4872, 4921, 4929, 4930, 4948, 4956, 4957	
\@@_rowcolor	1341, 3885	\c_@@_table_collect_begin_tl ..	181, 183, 201
\@@_rowcolor:n	3891, 3894	\c_@@_table_print_tl	184, 185, 203
\@@_rowcolor_tabular	1123, 4178	\l_@@_tabular_width_dim	
\@@_rowcolors	1342, 3981 248, 1042, 1044, 1638, 2453	
\@@_rowcolors_i:nmmn	4025, 4060	\l_@@_tabularnote_tl	334, 802, 827, 1968, 1977
\l_@@_rowcolors_restart_bool ...	3977, 4008	\g_@@_tabularnotes_seq	
\g_@@_rows_seq ..	2125, 2127, 2129, 2131, 2133 333, 373, 1983, 1989, 2005	
\l_@@_rows_tl			
..... 3896, 3911, 3941, 4027, 4091, 4116			

<code>\@@_test_hline_in_block:nnnn</code>	4363, 4365, 4491
<code>\@@_test_hline_in_stroken_block:nnnn</code> ..	4367, 4513
<code>\@@_test_if_cell_in_a_block:nn</code>	4572, 4590, 4608, 4628
<code>\@@_test_if_cell_in_block:nnnnnnn</code> ...	4634, 4636
<code>\@@_test_if_math_mode:</code>	251, 1435, 2405
<code>\@@_test_in_corner_h:</code>	4368, 4396
<code>\@@_test_in_corner_v:</code>	4243, 4271
<code>\@@_test_vline_in_block:nnnn</code>	4237, 4239, 4502
<code>\@@_test_vline_in_stroken_block:nnnn</code> ..	4241, 4524
<code>\l_@@_the_array_box</code> ..	1265, 1281, 1962, 1963
<code>\c_@@_tikz_loaded_bool</code>	27, 41, 1330, 2568, 4703
<code>\@@_true_c:</code>	202, 1660
<code>\l_@@_type_of_col_tl</code> ..	785, 786, 2434, 2436
<code>\c_@@_types_of_matrix_seq</code>	6265, 6266, 6271, 6275
<code>\@@_update_for_first_and_last_row:</code> ..	883, 947, 1219, 2329, 2374
<code>\@@_use_arraybox_with_notes:</code> ...	1507, 2024
<code>\@@_use_arraybox_with_notes_b:</code> .	1504, 2008
<code>\@@_use_arraybox_with_notes_c:</code>	1505, 1536, 1960, 2022, 2061
<code>\@@_vdottedline:n</code>	1882, 4699
<code>\@@_vdottedline_i:n</code>	4706, 4711, 4715
<code>\@@_vline:nn</code>	1738, 4220, 4346
<code>\@@_vline_i:nn</code>	2562, 4225, 4229
<code>\@@_vline_i_complete:nn</code>	2562, 4336
<code>\@@_vline_ii:nnnn</code> ...	4252, 4263, 4296, 4337
<code>\l_@@_vlines_clist</code>	318, 550, 562, 591, 1585, 1591, 1621, 1633, 1890, 1897, 2566, 4344, 4345
<code>\l_@@_vpos_of_block_tl</code>	309, 310, 4968, 4970, 5052, 5062, 5140, 5157, 5191, 5193
<code>\@@_w:</code>	1578, 1658
<code>\g_@@_width_first_col_dim</code>	288, 1428, 1498, 2170, 2330, 2331
<code>\g_@@_width_last_col_dim</code>	287, 1427, 1557, 2279, 2375, 2376
<code>\@@_write_aux_for_cell_nodes:</code> ..	2603, 2665
<code>\l_@@_x_final_dim</code>	282, 2927, 2976, 2985, 2986, 2989, 2992, 2993, 3144, 3160, 3168, 3172, 3176, 3178, 3183, 3185, 3219, 3228, 3236, 3276, 3284, 3323, 3338, 3347, 3381, 3433, 3449, 3850, 4686, 4695, 4721, 5921, 5937, 5938, 5944, 6041, 6058, 6075
<code>\l_@@_x_initial_dim</code>	280, 2922, 2954, 2963, 2964, 2967, 2970, 2971, 3144, 3159, 3160, 3167, 3172, 3176, 3178, 3180, 3183, 3185, 3210, 3228, 3236, 3276, 3284, 3320, 3337, 3347, 3381, 3433, 3447, 3449, 3467, 3469, 3847, 4679, 4693, 4720, 5920, 5930, 5931, 5941, 6032, 6057, 6065
<code>\l_@@_xdots_color_tl</code> ..	483, 497, 3010, 3049, 3130, 3131, 3199, 3251, 3303, 3717, 3792, 3809
<code>\l_@@_xdots_down_tl</code> ...	501, 3310, 3331, 3366
<code>\l_@@_xdots_line_style_tl</code>	459, 461, 493, 3295, 3303, 4696, 4726
<code>\l_@@_xdots_shorten_dim</code>	455, 456, 499, 2550, 3317, 3318, 3407, 3418, 3426
<code>\l_@@_xdots_up_tl</code>	502, 3309, 3330, 3356
<code>\l_@@_y_final_dim</code>	283, 2928, 3028, 3032, 3070, 3074, 3076, 3101, 3111, 3112, 3230, 3233, 3278, 3281, 3323, 3338, 3346, 3383, 3438, 3457, 3851, 4677, 4725, 5775, 5797, 5812, 5958, 5973, 5974, 5979, 5997, 6014, 6058, 6066, 6076
<code>\l_@@_y_initial_dim</code>	281, 2923, 3021, 3031, 3069, 3070, 3074, 3076, 3083, 3093, 3094, 3230, 3235, 3278, 3283, 3320, 3337, 3346, 3383, 3438, 3455, 3457, 3467, 3470, 3848, 4675, 4676, 4677, 4723, 5773, 5797, 5812, 5955, 5966, 5967, 5979, 5996, 6013, 6057, 6066, 6076
<code>\</code>	2116, 2138, 5639, 5645, 5651, 6181, 6182, 6225, 6234, 6262, 6322, 6327, 6332, 6337, 6342, 6380, 6385, 6391, 6398, 6405, 6411, 6412, 6427, 6435, 6441, 6447, 6448, 6459, 6471, 6478, 6484, 6491, 6498, 6507, 6513, 6520, 6527, 6533, 6539, 6551, 6555, 6560, 6566, 6575, 6576, 6590, 6591, 6607, 6611, 6612, 6630, 6631, 6673, 6674, 6725, 6726, 6776, 6777, 6830, 6831
<code>\{</code>	267, 1663, 1807, 1818, 1844, 2412, 5667, 6037, 6440, 6520, 6673, 6776
<code>\}</code>	267, 1666, 1807, 1818, 1829, 2412, 5667, 6046, 6440, 6520, 6673, 6776
<code>\ </code>	2414, 5666
<code>\sqcup</code> ..	6261, 6289, 6294, 6301, 6309, 6310, 6321, 6322, 6379, 6404, 6405, 6418, 6424, 6428, 6431, 6434, 6446, 6452, 6464, 6502, 6503, 6504, 6512, 6518, 6525, 6538, 6545, 6546, 6554
A	
<code>\A</code>	5865
<code>\aboverulesep</code>	1999
<code>\addtocounter</code>	390
<code>\alph</code>	336
<code>\anchor</code>	2662, 2663
<code>\arraybackslash</code>	1751
<code>\arraycolor</code>	1343, 6261
<code>\arraycolsep</code>	574, 576, 578, 1041, 1141, 1272, 1273, 1535, 1539, 4682, 4689
<code>\arrayrulecolor</code>	95
<code>\arrayrulewidth</code>	128, 133, 145, 510, 874, 1058, 1060, 1066, 1097, 1588, 1594, 1681, 1731, 1893, 1900, 2016, 2055, 2182, 2184, 2190, 2200, 2202, 2208, 2231, 2233, 2239, 2257, 2259, 2265, 4112, 4113, 4115, 4131, 4133, 4312, 4313, 4315, 4326, 4332, 4438, 4449, 4455, 4484, 4761, 5448, 5503, 5528, 5530, 5542, 5797, 5980, 5984
<code>\arraystretch</code>	1140, 3084, 3102, 5044, 5133, 5149, 5956, 5959
<code>\AtBeginDocument</code>	23, 28, 73, 89, 165, 189, 340, 454, 456, 458, 470, 654, 670, 2630, 3474, 3647, 3723, 3801, 3834, 4660
<code>\AutoNiceMatrix</code>	5668
<code>\AutoNiceMatrixWithDelims</code> ...	5628, 5660, 5672

B			
\baselineskip	98, 105	\box_move_up:nn	64, 66, 68, 991, 1957, 2022, 2061
\bgroup	1422	\box_rotate:Nn	902
\bigskip	1464, 2710	\box_set_dp:Nn	924, 943, 1914
\Block	1188, 6477, 6525, 6566	\box_set_ht:Nn	930, 941, 1913
\BNiceMatrix	5620	\box_set_wd:Nn	918
\bNiceMatrix	5617	\box_use:N	393, 909, 998
\Body	1289	\box_use_drop:N	949, 955, 974, 1780, 1916, 1957, 1958, 1963, 2336, 5116, 5398, 5432
bool commands:		\box_wd:N	394, 919, 946, 953, 1012, 1277, 1279, 1962, 2081, 2093, 2331, 2334, 2376, 2380, 5085, 5686
\bool_do_until:Nn	2752, 2820	\l_tmpa_box	376, 393, 394, 1276, 1277, 1278, 1279, 1522, 1913, 1914, 1916, 1957, 1958, 2074, 2087
\bool_gset_false:N	912, 958, 959, 1080, 1204, 1357, 1429, 2341, 2388, 4500, 4511, 4522, 4533, 5076	\l_tmpb_box	2067, 2081, 2082, 2093
\bool_gset_true:N	2175, 2305, 2350, 2441, 3492, 3508, 3524, 3548, 3571, 3582, 3798, 4235, 4361	C	
\bool_if:NTF	156, 173, 656, 661, 672, 677, 843, 846, 939, 1054, 1081, 1128, 1138, 1195, 1196, 1216, 1266, 1328, 1330, 1359, 1362, 1364, 1421, 1436, 1446, 1477, 1555, 1560, 1571, 1576, 1602, 1778, 1794, 1828, 1995, 2161, 2178, 2196, 2214, 2227, 2253, 2274, 2280, 2309, 2327, 2355, 2372, 2462, 2464, 2483, 2486, 2505, 2531, 2546, 2568, 2603, 3073, 3075, 3222, 3270, 3490, 3506, 3522, 3546, 3569, 4581, 4599, 4617, 4744, 4754, 4776, 5047, 5071, 5136, 5152, 5249, 5295, 5353, 5679, 6237, 6247, 6280, 6426	\c	210, 1606
\bool_if:nTF	191, 342, 369, 1017, 1487, 2906, 3823, 4075, 4770, 5403, 5776, 5786, 5788, 5801, 5807, 5816	\Cdots	1178, 3497, 3500
\bool_lazy_all:nTF	1617, 1629, 2555, 4493, 4504, 4515, 4526	\cdots	1113, 1170
\bool_lazy_and:nnTF	1685, 2217, 2310, 2517, 3062, 3329, 3631, 3988, 4016, 4086, 4306, 4431, 5466, 6006, 6023	\cellcolor	1122, 1338, 4173
\bool_lazy_or:nnTF	490, 951, 1009, 1610, 1945, 1966, 2044, 2358, 3139, 3387, 4067, 4099, 4103, 4153, 4157, 4573, 4591, 4609, 4988, 4993, 5013, 5909	\chessboardcolors	1345
\bool_lazy_or_p:nn	2313	\cline	148, 1175, 1176
\bool_not_p:n	1620, 1622, 1632, 1634, 2219, 2519	clist commands:	
\bool_set:Nn	3143, 4010	\clist_clear:N	4146
\c_false_bool	1839, 1847, 1860, 1866, 1872, 3486, 3502, 3518	\clist_if_empty:NTF	4242, 4368, 5259
\g_tmpa_bool	4235, 4244, 4278, 4286, 4291, 4361, 4369, 4403, 4411, 4416, 4500, 4511, 4522, 4533	\clist_if_in:NnTF	1089, 1591, 1897, 4345, 4468, 5563, 5565, 5567, 5569, 5588
\l_tmpb_bool	4578, 4592, 4610, 4632, 4645	\clist_if_in:nnTF	5549
\c_true_bool	1806, 1817	\clist_map_inline:Nn	4093, 4116, 4147, 4538, 5547, 6004, 6021
box commands:		\clist_map_inline:nn	2429, 3947, 4001
\box_clear_new:N	1130, 1265	\clist_new:N	303, 317, 318, 319, 320, 468
\box_dp:N	868, 888, 925, 944, 999, 1156, 1165, 1224, 1756, 1914, 2074, 2087, 3102, 5106, 5959	\clist_put_right:Nn	4164
\box_gclear_new:N	5035	\clist_set:Nn	562, 563, 590, 591, 592
\box_grotate:Nn	5073	\clist_set_eq:NN	4145
\box_ht:N	869, 890, 896, 908, 931, 942, 991, 1158, 1160, 1163, 1222, 1752, 1913, 2074, 2087, 3084, 5097, 5956	\l_tmpa_clist	4145, 4147
\box_move_down:nn	999	\CodeAfter	838, 1192, 2119, 2122, 2304, 2349, 2583, 6578
		\CodeBefore	1418, 6261
		\color	99, 106, 160, 162, 1526, 1544, 3004, 3007, 3010, 3043, 3046, 3049, 3124, 3127, 3131, 3199, 3251, 3711, 3714, 3717, 3786, 3789, 3792, 3809, 3873, 4034, 4035, 4046, 4047, 5042, 5177, 5806, 6138, 6162
		\colorlet	259, 260, 853, 860, 2319, 2363
		\columncolor	1124, 1344, 2592, 4196
		\cr	132, 150, 2298
		\crrcr	2155
		cs commands:	
		\cs_generate_variant:Nn	58, 152, 3326, 3866, 4766, 4767
		\cs_gset:Npn	99, 106, 2470, 2477, 2491, 2498, 4759
		\cs_gset_eq:NN	186, 220, 1149, 1438, 2606
		\cs_if_exist:NTF	57, 1131, 1134, 1232, 1239, 1250, 1257, 1400, 1439, 1442, 2652, 2653, 2674, 2787, 2800, 2855, 2868, 2957, 2979, 3087, 3105, 3679, 3697, 3754, 3772, 4746, 4799, 5315, 5333, 5781, 5926, 5933, 5962, 5969
		\cs_if_exist_p:N	491, 3991, 4019, 4575, 4594, 4612

<code>\cs_if_free:NTF</code>	216, 2999, 3038, 3119, 3194, 3246
<code>\cs_if_free_p:N</code>	3825, 3827
<code>\cs_new_protected:Npx</code>	2632, 3836, 4662
<code>\cs_set:Nn</code>	642, 645, 648
<code>\cs_set:Npn</code>	95, 96, 102, 103, 108, 120, 121, 135, 137, 138, 160, 162, 339, 1105, 2746, 2808, 2876, 3721, 3796, 4472, 4473, 4479, 4480, 5044, 5133, 5149
<code>\cs_set_nopar:Npn</code>	1031, 1043, 1140, 1143, 4941, 4942
<code>\cs_set_nopar:Npx</code>	1044
<code>\cs_set_protected:Npn</code>	5657
<code>\cs_set_protected_nopar:Npn</code>	5023, 5282
D	
<code>\Ddots</code>	1180, 3530, 3531, 3536, 3537
<code>\ddots</code>	1115, 1172
<code>\diagbox</code>	1193, 5023, 5282
dim commands:	
<code>\dim_add:Nn</code>	4910
<code>\dim_compare:nNnTF</code>	98, 105, 916, 922, 928, 1638, 2215, 2380, 2967, 2989, 3178, 4823, 4833, 5325, 5345, 5686
<code>\dim_gzero:N</code>	920, 926, 932
<code>\dim_max:nn</code>	4812, 4816
<code>\dim_min:nn</code>	4804, 4808
<code>\dim_ratio:nn</code>	3237, 3285, 3401, 3406, 3417, 3425, 3434, 3439, 3450, 3458
<code>\dim_set:Nn</code>	4785, 4792, 4803, 4807, 4811, 4815, 4827, 4828, 4837, 4838, 4882, 4895
<code>\dim_set_eq:NN</code>	4783, 4790, 4890, 4904
<code>\dim_sub:Nn</code>	4907
<code>\dim_use:N</code>	4824, 4834, 4885, 4886, 4898, 4899, 4922, 4923, 4924, 4925, 4949, 4950, 4951, 4952
<code>\dim_zero_new:N</code>	4782, 4784, 4789, 4791
<code>\c_max_dim</code> .	2954, 2967, 2976, 2989, 4783, 4785, 4790, 4792, 4824, 4834, 5312, 5325, 5330, 5345, 5777, 5778, 5920, 5921, 5941, 5944
<code>\l_tmpc_dim</code>	284, 2725, 2729, 4112, 4113, 4136, 4305, 4313, 4318, 4323, 4329, 4430, 4441, 4446, 4452, 5304, 5310, 5352, 5473, 5484, 5579, 5582, 5718, 5721, 5729
<code>\l_tmpd_dim</code>	285, 2727, 2730, 4133, 4136, 4314, 4318, 4437, 4441, 5306, 5310, 5330, 5341, 5345, 5348, 5352, 5482, 5485, 5720, 5721, 5733
<code>\dotfill</code>	1191, 5675
<code>\dots</code>	1117
<code>\doublerulesep</code>	1732, 4315, 4327, 4438, 4450, 4485
<code>\doublerulesepcolor</code>	102
<code>\draw</code>	3314
E	
<code>\egroup</code>	1570
else commands:	
<code>\else:</code>	253
<code>\endarray</code>	2107, 2135
<code>\endBNiceMatrix</code>	5621
<code>\endbNiceMatrix</code>	5618
<code>\endNiceArray</code>	2449, 2458
<code>\endNiceArrayWithDelims</code>	2398, 2408
<code>\endpgfscope</code>	2688, 2701, 3371, 5732
<code>\endpNiceMatrix</code>	5609
<code>\endsavenotes</code>	1571
<code>\endtabularnotes</code>	1990
<code>\endVNiceMatrix</code>	5615
<code>\endvNiceMatrix</code>	5612
<code>\enskip</code>	1804, 1837, 1845, 1858, 1864
<code>\ensuremath</code>	3491, 3507, 3523, 3547, 3570
<code>\everycr</code>	131, 150, 1153
exp commands:	
<code>\exp_after:wN</code>	198, 1463, 1527, 1545, 1599, 5986
<code>\exp_args:Ne</code>	3592
<code>\exp_args:NNc</code>	4748
<code>\exp_args:NNe</code>	3588
<code>\exp_args:Nne</code>	2436
<code>\exp_args:NNV</code>	2127, 3478, 3494, 3510, 3526, 3550, 3651, 3727, 3805
<code>\exp_args:NNv</code>	1455
<code>\exp_args:NNx</code>	1088, 1896
<code>\exp_args:No</code>	3302
<code>\exp_args:NV</code> ...	1580, 2103, 2130, 2132, 5457
<code>\exp_args:Nxx</code>	5016, 5017
<code>\exp_last_unbraced:NV</code>	1360, 2585, 5271
<code>\exp_not:N</code>	42, 43, 46, 47, 1681, 1725, 4184, 4196, 4664, 4665, 5052, 5062, 5140, 5157, 5274, 6091
<code>\exp_not:n</code>	1023, 2598, 3661, 3737, 4173, 4184, 4186, 4197, 5032, 5114, 5126, 5127, 5242, 5254, 5264, 5276, 5291, 5696, 5697
<code>\ExplSyntaxOff</code>	218, 2481, 2502, 2528, 2600, 4562, 4763
<code>\ExplSyntaxOn</code>	215, 2467, 2488, 2507, 2593, 4555, 4756
<code>\extrarowheight</code> ...	3084, 5045, 5134, 5150, 5956
F	
<code>\fi</code>	110, 1582, 4472, 4489
fi commands:	
<code>\fi:</code>	255
<code>\five</code>	2657, 2662
flag commands:	
<code>\flag_clear_new:n</code>	6084
<code>\flag_height:n</code>	6109
<code>\flag_raise:n</code>	6108
<code>\fontdimen</code>	1953
fp commands:	
<code>\fp_eval:n</code>	3342
<code>\fp_to_dim:n</code>	3377
<code>\futurelet</code>	115
G	
<code>\globaldefs</code>	1566, 5220
group commands:	
<code>\group_insert_after:N</code>	5680, 5681, 5683, 5684
H	
<code>\halign</code>	1167
<code>\hbox</code>	1052, 1531, 2022, 2061, 2180, 2198, 2225, 2229, 2255, 2683, 2696
hbox commands:	
<code>\hbox:n</code>	64, 66, 69
<code>\hbox_gset:Nn</code>	5037
<code>\hbox_overlap_left:n</code> ..	992, 1000, 2159, 2332
<code>\hbox_overlap_right:n</code>	393, 2276, 2378

`\hbox_set:Nn` 376, 989, 1276, 1278, 1522, 2067, 2082, 5294
`\hbox_set:Nw` 842, 1281, 1769, 2307, 2353
`\hbox_set_end:` .. 938, 1474, 1777, 2326, 2371
`\hbox_to_wd:nn` 418, 443
`\Hdotsfor` 1185, 6289, 6464
`\hdotsfor` 1118
`\hdottedline` 1182
`\heavyrulewidth` 2000
`\hfil` 1659
`\hfill` 128, 145
`\Hline` 1183, 4475
`\hline` 108
`\hrule` 112, 128, 145, 1097, 2000, 5811, 6143, 6167
`\hskip` 111
`\Hspace` 1184
`\hspace` 3583
`\hss` 1659

I

`\ialign` 1049, 1143, 1166, 5199
`\iddots` 1181, 3554, 3555, 3560, 3561
`\iddots` 59, 1116, 1173
 if commands:
`\if_mode_math:` 253
`\IfBooleanTF` 1465
`\ifnum` 110, 4472, 4489
`\ifstandalone` 1442
 int commands:
`\int_case:nnTF` 3528, 3534, 3552, 3558
`\int_compare:nNnTF` 123, 124, 140, 840, 841,
 850, 857, 893, 903, 1094, 1096, 1217, 1227,
 1246, 1303, 1305, 1475, 1479, 1490, 1494,
 1495, 1562, 1978, 2017, 2056, 2128, 2156,
 2356, 2762, 2769, 2773, 2775, 2830, 2837,
 2841, 2843, 3006, 3045, 3126, 3161, 3163,
 3611, 3625, 3713, 3788, 4062, 4107, 4125,
 4161, 4192, 4209, 4210, 4217, 4218, 4222,
 4638, 4640, 4642, 4644, 5041, 5078, 5090,
 5372, 5401, 5405, 5475, 5477, 5635, 5637,
 5639, 5643, 5645, 5647, 5649, 5651, 5990, 5992
`\int_compare_p:n`
 2908, 2909, 2910, 2911, 4077, 4079, 5467, 5468
`\int_do_until:nNnn` 4013
`\int_gadd:Nn` 3624
`\int_gincr:N` .. 839, 866, 1445, 1705, 1761,
 1783, 1789, 2251, 2351, 3224, 3272, 4741, 5022
`\int_if_even:nTF` 3956, 6109
`\int_if_odd_p:n` 4010
`\int_incr:N` 372, 1715
`\int_min:nn`
 2720, 2722, 2724, 2726, 2734, 2736, 2916, 2917
`\int_step_inline:nn`
 2718, 3952, 3954, 6003, 6020
`\int_step_inline:nnnn` 4570, 4588, 4603, 4606
`\l_tmpc_int` 4011, 4012, 4013
`\c_zero_int` 1795, 3859, 4068
 iow commands:
`\iow_now:Nn` 76, 213, 2507, 2508,
 2510, 2528, 2593, 2594, 2600, 4555, 4556, 4562
`\iow_shipout:Nn` 2467, 2468, 2475,
 2481, 2488, 2489, 2496, 2502, 4756, 4757, 4763
`\item` 1983, 1989

K

`\kern` 69
 keys commands:
`\keys_define:nn` 486, 506, 513,
 585, 632, 684, 691, 735, 777, 791, 808, 821,
 1367, 3574, 3963, 3972, 4730, 4960, 5169,
 5492, 5598, 5738, 5743, 5823, 5844, 5853, 6199
`\l_keys_key_str`
 6181, 6363, 6369, 6457, 6560,
 6565, 6575, 6590, 6611, 6629, 6672, 6724, 6775
`\keys_set:nn`
 516, 518, 532, 766, 769, 776, 1371, 1379,
 1459, 1460, 2435, 2445, 2454, 2609, 3009,
 3048, 3129, 3198, 3250, 3716, 3791, 3808,
 3967, 3986, 4743, 5236, 5745, 5749, 5876, 5915
`\keys_set_known:nn`
 3540, 3564, 5005, 5449, 5504, 5543
`\l_keys_value_tl` 6410, 6486, 6493, 6825

L

`\Ldots` 1177, 3481, 3484
`\ldots` 1112, 1169
`\leaders` 128, 145
`\left` 1527, 2070, 2085, 5807, 6139, 6163
 legacy commands:
`\legacy_if:nTF` 616
`\line` 2580, 6440, 6586

M

`\makebox` 1780
`\mathinner` 61
 mode commands:
`\mode_leave_vertical:` 1434, 1750
 msg commands:
`\msg_error:nn` 12
`\msg_error:nnn` 13
`\msg_error:nnnn` 14, 5217, 5224, 5228
`\msg_fatal:nn` 15
`\msg_fatal:nnn` 16
`\msg_new:nnn` 17
`\msg_new:nnnn` 18
`\msg_redirect_name:nnn` 20
`\multicolumn` 1187, 3585, 3637, 3643, 3664
`\multispan` 124, 125, 141, 142
`\myfiledate` 6
`\myfileversion` 7

N

`\newcolumntype` 225, 226, 227
`\newcounter` 332
`\NewDocumentCommand`
 .. 344, 367, 775, 1377, 2608, 3478, 3494,
 3510, 3526, 3550, 3651, 3727, 3805, 3885,
 3900, 3915, 3924, 3945, 3950, 3965, 3981,
 4167, 4178, 4190, 5437, 5628, 5668, 5883, 5896
`\NewDocumentEnvironment` 1417,
 2100, 2109, 2391, 2401, 2431, 2442, 2450, 4739
`\NewExpandableDocumentCommand` 231, 4975
`\newlist` 348, 359
`\NiceArray` 2447, 2456
`\NiceArrayWithDelims` 2396, 2406
 nicematrix commands:
`\g_nicematrix_code_after_tl` .. 269, 625,
 2124, 2585, 2587, 5239, 5251, 5261, 5756, 5763

`\g_nicematrix_code_before_tl`
 ... 1211, 2589, 2598, 4171, 4182, 4194, 5272
`\NiceMatrixLastEnv` 231
`\NiceMatrixOptions` 775, 6630, 6830
`\NiceMatrixoptions` 6490
`\noalign` 98, 105, 110, 133, 1076, 1149, 4472, 4652
`\nobreak` 362
`\normalbaselines` 1137
`\NotEmpty` 1194
`\nulldelimiterspace` 2081, 2093, 5792, 5982
`\nullfont` 5803, 5810, 6135, 6142, 6159, 6166
`\numexpr` 153, 154

O
`\omit` 123, 2158, 2174, 2250, 5753
`\OnlyMainNiceMatrix` 1190, 4201

P
`\par` 1977, 1985
peek commands:
`\peek_meaning:NTF` 158, 374, 1106
`\peek_meaning_ignore_spaces:NTF` 2102, 4475
`\peek_meaning_remove_ignore_spaces:NTF` 148
`\peek_remove_spaces:n` 3609, 4169, 4180, 4977
`\pgfdeclareshape` 2655
`\pgfextracty` 5375
`\pgfgetlastxy` 436
`\pgfpathcircle` 3466
`\pgfpathlineto` 4323, 4329, 4446, 4452,
 5524, 5535, 5582, 5595, 5721, 5997, 6014, 6048
`\pgfpathmoveto` 4322, 4328, 4445, 4451,
 5523, 5534, 5581, 5594, 5716, 5996, 6013, 6039
`\pgfpathrectanglecorners` 4135, 4316, 4439, 5483
`\pgfpointadd` 434
`\pgfpointanchor` 163, 234, 2678,
 2691, 2932, 2943, 2960, 2982, 3090, 3108,
 4802, 4810, 5320, 5338, 5357, 5359, 5376,
 5410, 5784, 5929, 5936, 5965, 5972, 6083, 6087
`\pgfpointdiff` 435, 1317, 1325, 1411, 1412
`\pgfpointlineattime` 3336
`\pgfpointorigin` 2165, 2288, 2663
`\pgfpointscale` 434
`\pgfpointshapeborder` 3846, 3849
`\pgfrememberpicturepositiononpagetrue` ...
 871, 965, 1064, 2164, 2188, 2206,
 2237, 2263, 2286, 2641, 2668, 2717, 3293,
 3845, 4298, 4423, 4673, 4718, 4845, 4855,
 4866, 5297, 5451, 5512, 5559, 5711, 5770, 5917
`\pgfscope` 2676, 2689, 3333, 5728
`\pgfset` 403, 428, 966, 5387, 5421, 5727, 5791, 5919
`\pgfsetbaseline` 964
`\pgfsetcornersarced` 4134, 5459
`\pgfsetlinewidth`
 4332, 4455, 5486, 5515, 5562, 5984
`\pgfsetrectcap` 4333, 4456
`\pgfsetroundcap` 5724
`\pgfsetstrokecolor` 5457
`\pgfsyspdfmark` 216, 217
`\pgftransformrotate` 3340
`\pgftransformshift` 409, 434, 2677, 2690, 2728,
 3334, 5386, 5408, 5729, 5733, 5793, 6062, 6072
`\pgfusepath`
 ... 3360, 3370, 3876, 4038, 4050, 4319, 5487
`\pgfusepathqfill` 3472, 4442

`\pgfusepathqstroke` 4334, 4457,
 5525, 5536, 5583, 5596, 5725, 5998, 6015, 6049
`\phantom` 3491, 3507, 3523, 3547, 3570
`\pNiceMatrix` 5608
prg commands:
`\prg_do_nothing:`
 177, 186, 192, 220, 503, 1149, 2583
`\prg_new_conditional:Nnn` 4065, 4073
`\prg_replicate:nn` 382, 2246,
 2247, 3664, 4324, 4447, 5638, 5641, 5644, 5650
`\prg_return_false:` 4070, 4082
`\prg_return_true:` 4071, 4081
`\ProcessKeysOptions` 6218
`\ProvideDocumentCommand` 59
`\ProvidesExplPackage` 4

Q
`\quad` 363
quark commands:
`\q_stop` 120, 121, 137, 138, 159, 161, 1360,
 1382, 1463, 1599, 1670, 1832, 1854, 2119,
 2122, 3799, 3813, 3814, 3880, 3935, 3940,
 4005, 4097, 4098, 4120, 4121, 4151, 4152,
 4939, 4946, 4980, 4981, 4985, 5271, 5442,
 5465, 5474, 5505, 5508, 5552, 5555, 5623,
 5632, 5885, 5890, 5903, 5906, 5986, 6094, 6102

R
`\rectanglecolor` 1339, 2591, 4184
`\refstepcounter` 391
regex commands:
`\regex_const:Nn` 210
`\regex_match:nnTF` 5865
`\regex_replace_all:NnN` 1604
`\relax` 153, 154
`\renewcommand` 196
`\RenewDocumentEnvironment`
 5607, 5610, 5613, 5616, 5619
`\RequirePackage` 1, 3, 9, 10, 11
`\right` 1545, 2077, 2089, 5816, 6147, 6171
`\rotate` 1189
`\roundedrectanglecolor` 1340, 5274
`\rowcolor` 1123, 1341
`\rowcolors` 1342

S
`\savedanchor` 2657
`\savenotes` 1421
scan commands:
`\scan_stop:` 2586
`\scriptstyle`
 846, 2309, 2355, 3321, 3322, 3356, 3366
seq commands:
`\seq_clear:N` 1597
`\seq_clear_new:N` 2509, 4537
`\seq_count:N` 2129, 3862
`\seq_gclear:N`
 ... 1215, 1448, 1449, 1450, 1451, 2005, 2584
`\seq_gclear_new:N` 1197, 1198, 1356, 2125, 2141
`\seq_gpop_left:NN` 2131, 2143
`\seq_gput_left:Nn` 621, 3613, 3615, 5234
`\seq_gput_right:Nn` 373, 1678, 2887,
 3616, 3861, 5111, 5123, 5246, 5699, 5871, 5891
`\seq_gset_from_clist:Nn` .. 2512, 2524, 4558

<code>\CT@everycr</code>	1147	2111, 2138, 3010, 3049, 3130, 3199, 3251,	
<code>\CT@row@color</code>	1149	3717, 3792, 3809, 4033, 4045, 5862, 6104, 6288	
<code>\if@tempswa</code>	1582	<code>\tl_if_empty_p:N</code>	1621, 1633, 3330, 3331
<code>\NC@</code>	1105	<code>\tl_if_empty_p:n</code>	1968
<code>\NC@find</code>	177, 205	<code>\tl_if_eq:NNTF</code>	3295
<code>\NC@list</code>	1582	<code>\tl_if_eq:NnTF</code>	1086, 1585, 1890,
<code>\NC@rewrite@S</code>	178, 196	1915, 2026, 4344, 4467, 4692, 4694, 6002, 6019	
<code>\new@ifnextchar</code>	1202	<code>\tl_if_eq:nNTF</code> ...	611, 753, 1832, 1854, 3858
<code>\newcol@</code>	1107, 1108	<code>\tl_if_exist:NNTF</code>	1452
<code>\nicematrix@redefine@check@rerun</code> ..	76, 79	<code>\tl_if_in:NnTF</code>	4004, 4096, 4119, 4150
<code>\pgf@relevantforpicturesizefalse</code>		<code>\tl_if_in:nNTF</code>	1807, 1818, 1829, 1844
.....	1310, 2642, 2669, 3294,	<code>\tl_if_single_token:nTF</code>	555, 761
3463, 3870, 4000, 4299, 4424, 4846, 4856,		<code>\tl_if_single_token_p:n</code>	58
4867, 5298, 5452, 5513, 5560, 5710, 5771, 5918		<code>\tl_item:Nn</code>	182, 183, 185
<code>\pgfsys@getposition</code>		<code>\tl_map_inline:nn</code>	2112
.....	1308, 1315, 1323, 1403, 1406	<code>\tl_new:N</code>	181,
<code>\pgfsys@markposition</code>		184, 239, 257, 269, 270, 271, 274, 278, 301,	
.....	994, 1002, 1059, 1307,	302, 305, 307, 309, 334, 459, 463, 481, 483, 484	
2162, 2183, 2201, 2232, 2258, 2282, 2684, 2697		<code>\tl_put_left:Nn</code>	1129, 1365
<code>\pgfutil@check@rerun</code>	81, 82	<code>\tl_put_right:Nn</code>	601, 1219, 1291, 1455
<code>\reserved@a</code>	115	<code>\tl_range:nnn</code>	84
<code>\rvtx@iff@format@geq</code>	57	<code>\tl_set:Nn</code>	182, 3941, 3942, 4006,
<code>\set@color</code>	5041, 5294	4027, 4106, 4108, 4124, 4126, 4160, 4162,	
<code>\tikz@library@external@loaded</code>	1439	4231, 5006, 5476, 5478, 5509, 5510, 5556, 5557	
tex commands:		<code>\tl_set_eq:NN</code>	461, 3938, 3939, 4109,
<code>\tex_mkern:D</code>	63, 65, 67, 70	4247, 4372, 4696, 4726, 5002, 5506, 5507,	
<code>\tex_the:D</code>	200	5553, 5554, 5888, 5889, 5904, 5905, 5907, 5908	
<code>\textfont</code>	1953	<code>\tl_set_rescan:Nnn</code>	
<code>\textit</code>	336	2126, 3477, 3650, 3726, 3804
<code>\textsuperscript</code>	337, 338	<code>\tl_to_str:n</code>	6272
<code>\the</code>	153, 154, 1582, 1599	<code>\g_tmpa_tl</code>	179, 182, 185
<code>\thetabularnote</code>	339	<code>\l_tmpc_tl</code>	3936, 3938, 3941,
<code>\tikzexternaldisable</code>	1441	4109, 4128, 4232, 4246, 4247, 4250, 4255,	
<code>\tikzset</code>	1332, 1443, 2570	4257, 4261, 4266, 4268, 4358, 4371, 4372,	
tl commands:		4375, 4380, 4382, 4386, 4391, 4393, 5506,	
<code>\tl_clear:N</code>	4257, 4268, 4382, 4393, 5447	5518, 5531, 5553, 5570, 5576, 5886, 5888, 5892	
<code>\tl_clear_new:N</code>	3936, 3937, 3984,	<code>\l_tmpd_tl</code>	3937, 3939, 3942, 5507,
4232, 4358, 5886, 5887, 5899, 5900, 5901, 5902		5520, 5529, 5554, 5566, 5587, 5887, 5889, 5892	
<code>\tl_const:Nn</code>		token commands:	
.....	42, 43, 46, 47, 460, 2300, 2345, 6095	<code>\token_to_str:N</code>	6260, 6261,
<code>\tl_count:n</code>	1934, 2038	6289, 6374, 6379, 6385, 6410, 6418, 6424,	
<code>\tl_gclear:N</code>	1584, 2582, 2587, 3875	6428, 6440, 6446, 6464, 6477, 6518, 6525,	
<code>\tl_gclear_new:N</code>		6538, 6555, 6565, 6577, 6586, 6592, 6630, 6830	
... 1205, 1206, 1207, 1208, 1209, 1210, 1211			
<code>\tl_gput_left:Nn</code>	1017, 1615, 4194		
<code>\tl_gput_right:Nn</code>			
.....	1017, 1627, 1680, 1723, 1737,		
1805, 1816, 1838, 1846, 1859, 1865, 1871,			
1881, 3653, 3729, 3864, 4171, 4182, 4487,			
4657, 5025, 5239, 5251, 5261, 5272, 5284, 5689			
<code>\tl_gset:Nn</code> 179, 183, 185, 1423, 1424, 1425,			
1587, 1593, 1799, 1800, 1835, 1861, 2596, 3862			
<code>\tl_if_blank:nTF</code>	3887, 3890, 3902,		
3905, 3917, 3920, 3926, 3929, 4031, 4043, 4979			
<code>\tl_if_blank_p:n</code>			
4100, 4104, 4154, 4158, 4308, 4433, 4989, 4994			
<code>\tl_if_empty:NNTF</code>	1084, 1462, 1525,		
1543, 1977, 2565, 2566, 2589, 4122, 4123,			
4246, 4250, 4261, 4371, 4375, 4386, 5000,			
5040, 5237, 5269, 5453, 5805, 5985, 6137, 6161			
<code>\tl_if_empty:nTF</code>			
.....	599, 737, 779, 795, 811, 829,		

U

`\unskip`

use commands:

<code>\use:N</code>	1020, 1235, 1242, 1253,
1260, 1286, 1287, 1469, 1470, 2418, 2438, 3874	
<code>\use:n</code> 3810, 4201, 5050, 5060, 5138, 5155, 6090	
<code>\usepackage</code>	6244, 6254
<code>\usepgfmodule</code>	2

V

vbox commands:

<code>\vbox:n</code>	69
<code>\vbox_set_top:Nn</code>	905
<code>\vbox_to_ht:nn</code>	414, 441, 2073, 2086
<code>\vbox_to_zero:n</code>	907
<code>\vcenter</code>	1528, 2071, 5808, 6140, 6164, 6555
<code>\Vdots</code>	1179, 3513, 3516
<code>\vdots</code>	1114, 1171
<code>\Vdotsfor</code>	1186
<code>\vfill</code>	417, 443

<code>\vline</code>	114			X	
<code>\VNiceMatrix</code>	5614	<code>\xglobal</code>	853, 860, 2319, 2363		
<code>\vNiceMatrix</code>	5611				
<code>\vrule</code>	112, 1752, 1756			Z	
<code>\vskip</code>	111				
<code>\vtop</code>	1056	<code>\Z</code>	5865		

Contents

1	The environments of this package	2
2	The vertical space between the rows	2
3	The vertical position of the arrays	3
4	The blocks	4
4.1	General case	4
4.2	The mono-column blocks	5
4.3	The mono-row blocks	6
4.4	The mono-cell blocks	6
4.5	A small remark	6
5	The rules	7
5.1	Some differences with the classical environments	7
5.1.1	The vertical rules	7
5.1.2	The command <code>\cline</code>	8
5.2	The thickness and the color of the rules	8
5.3	The tools of nicematrix for the rules	8
5.3.1	The keys <code>hlines</code> and <code>vlines</code>	9
5.3.2	The key <code>hvlines</code>	9
5.3.3	The (empty) corners	9
5.4	The command <code>\diagbox</code>	10
5.5	Dotted rules	11
6	The color of the rows and columns	11
6.1	Use of <code>colortbl</code>	11
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	12
6.3	Color tools with the syntax of <code>colortbl</code>	15
7	The width of the columns	16
8	The exterior rows and columns	17
9	The continuous dotted lines	18
9.1	The option <code>nullify-dots</code>	20
9.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	20
9.3	How to generate the continuous dotted lines transparently	21
9.4	The labels of the dotted lines	22
9.5	Customisation of the dotted lines	22
9.6	The dotted lines and the rules	23
10	The <code>\CodeAfter</code>	24
10.1	The command <code>\line</code> in the <code>\CodeAfter</code>	24
10.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	24

11	The notes in the tabulars	26
11.1	The footnotes	26
11.2	The notes of tabular	27
11.3	Customisation of the tabular notes	28
11.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	30
12	Other features	30
12.1	Use of the column type <code>S</code> of <code>siunitx</code>	30
12.2	Alignment option in <code>{NiceMatrix}</code>	31
12.3	The command <code>\rotate</code>	31
12.4	The option <code>small</code>	31
12.5	The counters <code>iRow</code> and <code>jCol</code>	32
12.6	The option <code>light-syntax</code>	33
12.7	Color of the delimiters	33
12.8	The environment <code>{NiceArrayWithDelims}</code>	33
13	Use of Tikz with <code>nicematrix</code>	33
13.1	The nodes corresponding to the contents of the cells	33
13.2	The “medium nodes” and the “large nodes”	34
13.3	The nodes which indicate the position of the rules	36
13.4	The nodes corresponding to the command <code>\SubMatrix</code>	37
14	API for the developpers	37
15	Technical remarks	38
15.1	Definition of new column types	38
15.2	Diagonal lines	38
15.3	The “empty” cells	39
15.4	The option <code>exterior-arraycolsep</code>	39
15.5	Incompatibilities	39
16	Examples	40
16.1	Notes in the tabulars	40
16.2	Dotted lines	41
16.3	Dotted lines which are no longer dotted	42
16.4	Stacks of matrices	43
16.5	How to highlight cells of a matrix	45
16.6	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	48
17	Implementation	48
18	History	202
	Index	209