

The package `nicematrix`*

F. Pantigny
 fpantigny@wanadoo.fr

March 10, 2021

Abstract

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

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

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

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

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**.

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 5.12 of `nicematrix`, at the date of 2021/03/10.

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>

1 The environments of this package

The package `nicematrix` defines the following new environments.

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

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

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

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

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

Important

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

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

2 The vertical space between the rows

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

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

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

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`.²

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

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

²One should remark that these parameters apply also to the columns of type `S` of `siunitx` whereas the package `cellspace` is not able to act on such columns of type `S`.

3 The vertical position of the arrays

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

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

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

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

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

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

1. an item

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

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

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

1. an item

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

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

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

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

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

4 The blocks

4.1 General case

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

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

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

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

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

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

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

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

One may wish to raise the size of the “A” placed in the block of the previous example. Since this element is composed in math mode, it's not possible to use directly a command like `\large`, `\Large` and `\LARGE`. That's why the command `\Block` provides an option between angle brackets to specify some TeX code which will be inserted before the beginning of the math mode.³

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

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

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

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

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

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

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;
- the key `fill` takes in as value a color and fills the block with that color;

³This 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 `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);
- **New 5.12** 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 pt⁴).

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=red!15,rounded-corners]{2-2}{\LARGE De très jolies fleurs}
& & souci \\
pervenche & & lys \\
arum      & iris & jacinthe & muguet
\end{NiceTabular}
```

rose	tulipe	marguerite	dahlia
violette	De très jolies fleurs		souci
pervenche			lys
arum	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).

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

⁴This value is the initial value of the *rounded corners* of Tikz.

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`.⁵

```
\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.⁶

4.5 A small remark

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

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

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

⁵If 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).

⁶One may consider that the default value of the first mandatory argument of `\Block` is 1-1.

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.

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. 35):

```
\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.

```
\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 35.

5.3 The keys `hlines` and `vlines`

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

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

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

5.4 The key `hvlines`

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

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

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

5.5 The key `hvlines-except-corners`

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

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

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

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

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

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

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

5.6 The command `\diagbox`

The command `\diagbox` (inspired by the package `diagbox`), allows, when it is used in a cell, to slash that cell diagonally downwards.⁷

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

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

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

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

5.7 Dotted rules

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

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

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

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

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

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

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

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

6 The color of the rows and columns

6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

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

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

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

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

6.2 The tools of nicematrix in the code-before

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

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

New 5.12 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` and `\chessboardcolors`.⁹

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

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

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

```
\begin{NiceTabular}{|c|c|c|}
\CodeBefore
  \cellcolor{red!15}{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

- 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).

⁹Remark that, in the `code-before`, some PGF/Tikz nodes corresponding to the position to the potential rules are available: cf. p. 33.

```

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

```

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`¹⁰. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows of the tabular with the two colors (provided in second and third argument), beginning with the row whose number is given in first (mandatory) argument.

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

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

- The key `cols` describes a set of columns. The command `\rowcolors` will color only the cells of these columns. The value is a comma-separated list of intervals of the form *i-j*.
- With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.¹¹
- 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).

```

 $\backslash\begin{NiceTabular}{c l r}[hvlines]$ 
 $\backslash\CodeBefore$ 
 $\backslash\rowcolors{2}{blue!10}{}[cols=2-3,restart]$ 
 $\backslash\Body$ 
 $\backslash\Block{1-*}{Results} \ \backslash\backslash$ 
 $John \ \& \ 12 \ \backslash\backslash$ 
 $Stephen \ \& \ 8 \ \backslash\backslash$ 
 $Sarah \ \& \ 18 \ \backslash\backslash$ 
 $Ashley \ \& \ 20 \ \backslash\backslash$ 
 $Henry \ \& \ 14 \ \backslash\backslash$ 
 $Madison \ \& \ 15$ 
 $\backslash\end{NiceTabular}$ 

```

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

¹⁰The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

¹¹Otherwise, the color of a given row relies only upon the parity of its number.

```

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

```

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

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

```

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

```

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

We have used the key `r` which aligns all the columns rightwards (cf. p. 28).

One should remark that these 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.¹²

¹²As for now, this key is *not* available in `\NiceMatrixOptions`.

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

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.¹³

```
\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`¹⁴. The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

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

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

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

Several compilations may be necessary to achieve the job.

8 The exterior rows and columns

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

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

```
\begin{pNiceMatrix}[first-row,last-row,first-col,last-col,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}
```

¹³The 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).

¹⁴At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

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

The dotted lines have been drawn with the tools presented p. 17.

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.¹⁵
- 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. 30) is used, `nicematrix` has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (but not the number of columns).
 - In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

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

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

```

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

```

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

¹⁵The users wishing exteriors columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 22).

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 35.
- A specification of color present in `code-for-first-row` also applies to a dotted line draw in this exterior “first row” (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 14) doesn't apply to the “first column” and “last column”.
- For technical reasons, it's not possible to use the option of the command `\` after the “first row” or before the “last row”. The placement of the delimiters would be wrong. If you are looking for a workaround, consider the command `\SubMatrix` in the `\CodeAfter` described p. 22.

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`.¹⁶

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells¹⁷ 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`.¹⁸

```
\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}
```

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

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

```
\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}
```

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

¹⁶The 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`.

¹⁷The precise definition of a “non-empty cell” is given below (cf. p. 36).

¹⁸It's also possible to change the color of all theses dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 20.

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

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

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.¹⁹

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}
```

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

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.

¹⁹In `nicematrix`, one should use `\hspace*` and not `\hspace` for such an usage because `nicematrix` loads `array`. One may also remark that it's possible to fix the width of a column by using the environment `{NiceArray}` (or one of its variants) with a column of type `w` or `W`: see p. 14

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

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

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

```

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

```

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

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

```

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

```

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`²⁰ is loaded (but you might have problem if you use `\rowcolor` on the same row as `\Hdotsfor`).

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

```

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

```

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

²⁰We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

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`.²¹

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`¹⁶ 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}
\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. 22) accept two optional arguments specified by the tokens `_` and `^` for labels positionned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

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

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

9.5 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. 22) 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`;

²¹The 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.

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

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

The option `xdots/color`

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

The option `xdots/shorten`

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

The option `xdots/line-style`

It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).²²

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

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

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

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

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

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

9.6 The dotted lines and the rules

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

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

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

²²The 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.

10 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed after the construction of the matrix.²³

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 use, for instance, the PGF/Tikz nodes created by `nicematrix` in the `\CodeAfter`. These nodes are described further beginning on p. 31.

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. It 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}
```

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

The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 20).

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

New 5.10 The command `\SubMatrix` provides a way to put delimiters on a portion of the array considered as a submatrix. The command `\SubMatrix` takes in five arguments:

- the first argument is the left delimiter, which may be any extensible delimiter provided by LaTeX : `(`, `[`, `\{`, `\langle`, `\lgroup`, `\lfloor`, etc. but also the null delimiter `.`;
- the second argument is the upper-left corner of the submatrix with the syntax $i-j$ where i the number of row and j the number of column;
- the third argument is the lower-right corner with the same syntax;
- the fourth argument is the right delimiter;
- the last argument, which is optional, is a list of key-value pairs.²⁴

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.

²³There is also a key `code-before` described p. 11.

²⁴There 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}]`).

```

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

```

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

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).

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.

New 5.12 There is also the following keys, which are available only for an individual command `\SubMatrix`:

- **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.

```

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

```

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

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

```

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

```

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

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. 34.

11 The notes in the tabulars

11.1 The footnotes

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

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

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

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

11.2 The notes of tabular

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

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

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

^a Achard is an old family of the Poitou.

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

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

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

11.3 Customisation of the tabular notes

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

- `notes/para`
- `notes/bottomrule`
- `notes/style`
- `notes/label-in-tabular`
- `notes/label-in-list`

- `notes/enumitem-keys`
- `notes/enumitem-keys-para`
- `notes/code-before`

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

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

We detail these keys.

- The key `notes/para` requires the composition of the notes (at the end of the tabular) in a single paragraph.
Initial value: `false`
That key is also available within a given environment.
- The key `notes/bottomrule` adds a `\bottomrule` of `booktabs` *after* the notes. Of course, that rule is drawn only if there is really notes in the tabular. The package `booktabs` must have been loaded (before or after the package `nicematrix`). If it is not, an error is raised.
Initial value: `false`
That key is also available within a given environment.
- The key `notes/style` is a command whose argument is specified by `#1` and which gives the style of numerotation of the notes. That style will be used by `\ref` when referencing a tabular note marked with a command `\label`. The labels formatted by that style are used, separated by commas, when the user puts several consecutive commands `\tabularnote`. The marker `#1` is meant to be the name of a LaTeX counter.

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.

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. 26).

- 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. 37.

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).

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

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

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 }
$A = \begin{pNiceMatrix}[first-row,last-col=4]
e_1 & e_2 & e_3 & \\
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3
\end{pNiceMatrix}$
```

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

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

```

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

image of e_1 e_2 e_3

12.4 The option small

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

```

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

```

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

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 column²⁵. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `code-before` (cf. p. 11) and in the `\CodeAfter` (cf. p. 22), `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).

²⁵We recall that the exterior “first row” (if it exists) has the number 0 and that the exterior “first column” (if it exists) has also the number 0.

```

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

```

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

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

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

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

```

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

```

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

12.6 The option `light-syntax`

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

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

```

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

```

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

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

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.²⁶

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`.²⁷

```

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

```

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

²⁶The 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.

²⁷`delimiters-color` is a synonymous (deprecated) for `delimiters/color`.

12.8 The environment `{NiceArrayWithDelims}`

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

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

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

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`. ²⁸

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

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

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

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

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

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

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

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

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

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

²⁸One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 17).

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 43).

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

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`.²⁹

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

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

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

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.³⁰

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

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.³¹

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

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

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

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

²⁹There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

³⁰There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 15).

³¹The options `left-margin` and `right-margin` take dimensions as values but, if no value is given, the default value is used, which is `\arraycolsep` (by default: 5 pt). There is also an option `margin` to fix both `left-margin` and `right-margin` to the same value.

Here is an array composed with the following code:

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

13.3 The nodes which indicate the position of the rules

New 5.11 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 j (more specifically the potential position of those rules because maybe there are not actually drawn). These nodes are available in the `code-before` and the `\CodeAfter`.

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

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 `code-before`, 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	

13.4 The nodes corresponding to the command `\SubMatrix`

New 5.10 The command `\SubMatrix` available in the `\CodeAfter` has been described p. 22.

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 public³²:

- `\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 { ! O { black } }
```

³²According 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.

```

{
  \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

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 job³³:

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

The heavy vertical rule won't extend in the exterior rows.³⁴

```

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

```

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

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

15.2 Diagonal lines

By default, all the diagonal lines³⁵ 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):

```

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

```

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

³³The command `\vrule` is a TeX (and not LaTeX) command.

³⁴Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

³⁵We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

```

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

```

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

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

The same example without parallelization:

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

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

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 idea³⁶. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`³⁷. The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` of `array` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

³⁶In 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).*

³⁷And 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

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=;}
```

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.

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. 24.

Let’s consider that we wish to number the notes of a tabular with stars.³⁸

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument ³⁹

```
\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
    }
  }
}
```

³⁸Of course, it’s realistic only when there is very few notes in the tabular.

³⁹In fact: the value of its argument.

```

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

```

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

*Achard is an old family of the Poitou.

**The name Lefebvre is an alteration of the name Lefebure.

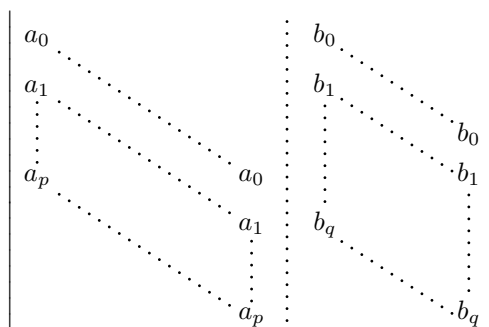
16.2 Dotted lines

An example with the resultant of two polynoms:

```

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

```



An example for a linear system:

```

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

```

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

16.3 Dotted lines which are no longer dotted

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

```

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

```

$$\left(\begin{array}{ccc|ccc} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ \hline & & 0 & & 1 & \\ & & & \ddots & & \\ & & & & 1 & \\ \hline & & 1 & & 0 & \\ & & & \ddots & & \\ & & & & 1 & \end{array} \right) \begin{array}{l} \leftarrow i \\ \leftarrow j \end{array}$$

$\uparrow i \qquad \uparrow j$

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}}
$\begin{pNiceMatrix}[first-row,first-col]
  & & \Ldots[line-style={solid,<->},shorten=Opt]^{n \text{ columns}} \\
  & 1 & 1 & 1 & \Ldots & 1 \\
  & 1 & 1 & 1 & & 1 \\
\vdots[line-style={solid,<->}]_{n \text{ rows}} & 1 & 1 & 1 & & 1 \\
  & 1 & 1 & 1 & & 1 \\
  & 1 & 1 & 1 & \Ldots & 1
\end{pNiceMatrix}$

```

$$\begin{array}{c}
 \xrightarrow{n \text{ columns}} \\
 \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix} \\
 \xleftarrow{n \text{ rows}}
 \end{array}$$

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}
```

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

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

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

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

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

New 5.12 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}
```

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

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

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

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

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

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

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

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

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

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

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 block⁴⁰).

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

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

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don’t spread the cells of the array. We recall that, on the other side, the command `\hline`, the specifier “|” and the options `hlines`, `vlines` and `hvlines` spread the cells.⁴¹

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.⁴²

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}
```

⁴⁰We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

⁴¹For the command `\cline`, see the remark p. 7.

⁴²In Overleaf, the “built-in” PDF viewer does not show transparency. You can switch to the “native” viewer in that case.

```

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

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

```

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

```

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

```

We obtain the following matrix.

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

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

```

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

```

We obtain the following matrix.

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

16.6 Utilisation of \SubMatrix in the code-before

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.

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

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_pgfutil@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_pgfutil@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`⁴³

```

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 { }
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: }

```

⁴³See question 99041 on TeX StackExchange.

```

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_pgfpintanchor \pgfpintanchor

```

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

```

237 \tl_new:N \l_@@_cell_type_tl
238 \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`.

```

239 \dim_new:N \g_@@_blocks_wd_dim

```

Idem pour the mono-row blocks.

```
240 \dim_new:N \g_@@_blocks_ht_dim
241 \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`.

```
242 \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.

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

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

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

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

```
245 \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.

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

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

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

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

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

```
249 \cs_new_protected:Npn \@@_test_if_math_mode:
250 {
251   \if_mode_math: \else:
252     \@@_fatal:n { Outside-math-mode }
253   \fi:
254 }
```

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

```
255 \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.

```
256 \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”.

```
257 \colorlet { nicematrix-last-col } { . }
258 \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`).

```
259 \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*.

```
260 \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).

```
261 \cs_new:Npn \@@_full_name_env:
262 {
263   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
264   { command \space \c_backslash_str \g_@@_name_env_str }
265   { environment \space \{ \g_@@_name_env_str \} }
266 }
```

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`).

```
267 \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.

```
268 \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.

```
269 \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.

```
270 \int_new:N \l_@@_old_iRow_int
271 \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.

```
272 \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.

```
273 \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).

```
274 \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).

```
275 \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`).

```
276 \tl_new:N \l_@@_code_before_tl
277 \bool_new:N \l_@@_code_before_bool
```

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

```
278 \dim_new:N \l_@@_x_initial_dim
279 \dim_new:N \l_@@_y_initial_dim
280 \dim_new:N \l_@@_x_final_dim
281 \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`).

```
282 \dim_zero_new:N \l_tmpc_dim
283 \dim_zero_new:N \l_tmpd_dim
```

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

```
284 \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”.

```
285 \dim_new:N \g_@@_width_last_col_dim
286 \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.

```
287 \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}`.

```
288 \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}`.

```
289 \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.

```
290 \seq_new:N \g_@@_pos_of_stroken_blocks_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`).

```
291 \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`).

```
292 \int_new:N \l_@@_row_min_int
293 \int_new:N \l_@@_row_max_int
294 \int_new:N \l_@@_col_min_int
295 \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.

```
296 \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).

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

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

```
298 \tl_new:N \l_@@_fill_tl
299 \tl_new:N \l_@@_draw_tl
300 \dim_new:N \l_@@_rounded_corners_dim
```

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).

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

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

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

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

```
303 \tl_new:N \l_@@_pos_of_block_tl
304 \tl_set:Nn \l_@@_pos_of_block_tl { c }
```

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

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

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

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

307 \dim_new:N \l_@@_submatrix_extra_height_dim
308 \dim_new:N \l_@@_submatrix_left_xshift_dim
309 \dim_new:N \l_@@_submatrix_right_xshift_dim
310 \clist_new:N \l_@@_hlines_clist
311 \clist_new:N \l_@@_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.

```
312 \int_new:N \l_@@_first_row_int
313 \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.

```
314 \int_new:N \l_@@_first_col_int
315 \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).

```
316 \int_new:N \l_@@_last_row_int
317 \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”.⁴⁴

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

Idem for `\l_@@_last_col_without_value_bool`

```
319 \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.

```
320 \int_new:N \l_@@_last_col_int
321 \int_set:Nn \l_@@_last_col_int { -2 }
```

However, we have also a boolean. Consider the following code:

```
\begin{pNiceArray}{cc}[last-col]
1 & 2 \\
3 & 4
\end{pNiceArray}
```

In such a code, the “last column” specified by the key `last-col` is not used. We want to be able to detect such a situation and we create a boolean for that job.

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

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

⁴⁴We can’t use `\l_@@_last_row_int` for this usage because, if `nicematrix` has read its value from the `aux` file, the value of the counter won’t be `-1` any longer.

The 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.

```
323 \newcounter { tablarnote }
```

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

```
324 \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.

```
325 \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).

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

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

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

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

```
328 \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`.

```
329 \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`.

```
330 \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 `tablarnotes` in the general case and a list `tablarnotes*` 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`).

```
331 \AtBeginDocument
332 {
333   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
334   {
335     \NewDocumentCommand \tablarnote { m }
336     { \@@_error:n { enumitem-not-loaded } }
337   }
338   {
```

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

```
339   \newlist { tablarnotes } { enumerate } { 1 }
340   \setlist [ tablarnotes ]
341   {
342     topsep = Opt ,
343     noitemsep ,
344     leftmargin = * ,
```

```

345         align = left ,
346         labelsep = Opt ,
347         label =
348             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
349     }
350     \newlist { tabularnotes* } { enumerate* } { 1 }
351     \setlist [ tabularnotes* ]
352     {
353         afterlabel = \nobreak ,
354         itemjoin = \quad ,
355         label =
356             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
357     }

```

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.⁴⁵

```

358     \NewDocumentCommand \tabularnote { m }
359     {
360         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
361         { \@@_error:n { tabularnote~forbidden } }
362         {

```

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

```

363         \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`.

```

364         \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
365         \peek_meaning:NF \tabularnote
366         {

```

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.

```

367         \hbox_set:Nn \l_tmpa_box
368         {

```

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

```

369         \@@_notes_label_in_tabular:n
370         {
371             \stepcounter { tabularnote }
372             \@@_notes_style:n { tabularnote }
373             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
374             {
375                 ,
376                 \stepcounter { tabularnote }
377                 \@@_notes_style:n { tabularnote }
378             }
379         }
380     }

```

⁴⁵We should try to find a solution to that problem.

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.

```

381         \addtocounter { tabularnote } { -1 }
382         \refstepcounter { tabularnote }
383         \int_zero:N \l_@@_number_of_notes_int
384         \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).

```

385         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
386     }
387 }
388 }
389 }
390 }

```

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.

```

391 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
392 {
393     \begin { pgfscope }
394     \pgfset
395     {
396         outer~sep = \c_zero_dim ,
397         inner~sep = \c_zero_dim ,
398         minimum~size = \c_zero_dim
399     }
400     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
401     \pgfnode
402     { rectangle }
403     { center }
404     {
405         \vbox_to_ht:nn
406         { \dim_abs:n { #5 - #3 } }
407         {
408             \vfill
409             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
410         }
411     }
412     { #1 }
413     { }
414     \end { pgfscope }
415 }

```

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

```

416 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
417 {
418     \begin { pgfscope }
419     \pgfset
420     {
421         outer~sep = \c_zero_dim ,
422         inner~sep = \c_zero_dim ,
423         minimum~size = \c_zero_dim
424     }
425     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }

```

```

426 \pgfpointdiff { #3 } { #2 }
427 \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
428 \pgfnode
429 { rectangle }
430 { center }
431 {
432   \vbox_to_ht:nn
433   { \dim_abs:n \l_tmpb_dim }
434   { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
435 }
436 { #1 }
437 { }
438 \end { pgfscope }
439 }

```

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.

```

440 \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`.

```

441 \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`).

```

442 \dim_new:N \l_@@_cell_space_top_limit_dim
443 \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.

```

444 \dim_new:N \l_@@_inter_dots_dim
445 \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).

```

446 \dim_new:N \l_@@_xdots_shorten_dim
447 \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.

```

448 \dim_new:N \l_@@_radius_dim
449 \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.

```
450 \tl_new:N \l_@@_xdots_line_style_tl
451 \tl_const:Nn \c_@@_standard_tl { standard }
452 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
453 \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).

```
454 \tl_new:N \l_@@_baseline_tl
455 \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`).

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

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

```
457 \bool_new:N \l_@@_parallelize_diags_bool
458 \bool_set_true:N \l_@@_parallelize_diags_bool
```

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

```
459 \clist_new:N \l_@@_except_corners_clist
460 \dim_new:N \l_@@_notes_above_space_dim
461 \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.).

```
462 \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).

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

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

```
464 \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”.

```
465 \bool_new:N \l_@@_medium_nodes_bool
466 \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`).

```
467 \dim_new:N \l_@@_left_margin_dim
468 \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`.

```
469 \dim_new:N \l_@@_extra_left_margin_dim
470 \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.

```
471 \tl_new:N \l_@@_end_of_row_tl
472 \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 “:”.

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

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

```
474 \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.

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

```
476 \keys_define:nn { NiceMatrix / xdots }
477 {
478   line-style .code:n =
479   {
480     \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.

```
481   { \cs_if_exist_p:N \tikzpicture }
482   { \str_if_eq_p:nn { #1 } { standard } }
483   { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
484   { \@@_error:n { bad-option-for~line-style } }
485   } ,
486   line-style .value_required:n = true ,
487   color .tl_set:N = \l_@@_xdots_color_tl ,
488   color .value_required:n = true ,
489   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
490   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 `_`.

```
491   down .tl_set:N = \l_@@_xdots_down_tl ,
492   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).

```
493   draw-first .code:n = \prg_do_nothing: ,
494   unknown .code:n = \@@_error:n { Unknown-key-for~xdots }
495 }
```

```

496 \keys_define:nn { NiceMatrix / rules }
497 {
498   color .tl_set:N = \l_@@_rules_color_tl ,
499   color .value_required:n = true ,
500   width .dim_set:N = \arrayrulewidth ,
501   width .value_required:n = true
502 }

```

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

```

503 \keys_define:nn { NiceMatrix / Global }
504 {
505   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
506   rules .value_required:n = true ,
507   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
508   standard-cline .default:n = true ,
509   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
510   cell-space-top-limit .value_required:n = true ,
511   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
512   cell-space-bottom-limit .value_required:n = true ,
513   cell-space-limits .meta:n =
514   {
515     cell-space-top-limit = #1 ,
516     cell-space-bottom-limit = #1 ,
517   } ,
518   cell-space-limits .value_required:n = true ,
519   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
520   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
521   light-syntax .default:n = true ,
522   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
523   end-of-row .value_required:n = true ,
524   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
525   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
526   last-row .int_set:N = \l_@@_last_row_int ,
527   last-row .default:n = -1 ,
528   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
529   code-for-first-col .value_required:n = true ,
530   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
531   code-for-last-col .value_required:n = true ,
532   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
533   code-for-first-row .value_required:n = true ,
534   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
535   code-for-last-row .value_required:n = true ,
536   hlines .clist_set:N = \l_@@_hlines_clist ,
537   vlines .clist_set:N = \l_@@_vlines_clist ,
538   hlines .default:n = all ,
539   vlines .default:n = all ,
540   vlines-in-sub-matrix .code:n =
541   {
542     \tl_if_single_token:nTF { #1 }
543     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
544     { \@@_error:n { One-letter-allowed } }
545   } ,
546   vlines-in-sub-matrix .value_required:n = true ,
547   hvlines .code:n =
548   {
549     \clist_set:Nn \l_@@_vlines_clist { all }
550     \clist_set:Nn \l_@@_hlines_clist { all }
551   } ,
552   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.

```

553   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
554   renew-dots .value_forbidden:n = true ,
555   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
556   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
557   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
558   create-extra-nodes .meta:n =
559     { create-medium-nodes , create-large-nodes } ,
560   left-margin .dim_set:N = \l_@@_left_margin_dim ,
561   left-margin .default:n = \arraycolsep ,
562   right-margin .dim_set:N = \l_@@_right_margin_dim ,
563   right-margin .default:n = \arraycolsep ,
564   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
565   margin .default:n = \arraycolsep ,
566   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
567   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
568   extra-margin .meta:n =
569     { extra-left-margin = #1 , extra-right-margin = #1 } ,
570   extra-margin .value_required:n = true ,
571 }

```

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

```

572 \keys_define:nn { NiceMatrix / Env }
573 {
574   delimiters/max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
575   except-corners .clist_set:N = \l_@@_except_corners_clist ,
576   except-corners .default:n = { NW , SW , NE , SE } ,
577   hvlines-except-corners .code:n =
578     {
579       \clist_set:Nn \l_@@_except_corners_clist { #1 }
580       \clist_set:Nn \l_@@_vlines_clist { all }
581       \clist_set:Nn \l_@@_hlines_clist { all }
582     } ,
583   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
584   code-before .code:n =
585     {
586       \tl_if_empty:nF { #1 }
587       {
588         \tl_put_right:Nn \l_@@_code_before_tl { #1 }
589         \bool_set_true:N \l_@@_code_before_bool
590       }
591     } ,

```

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

```

592   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
593   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
594   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
595   baseline .tl_set:N = \l_@@_baseline_tl ,
596   baseline .value_required:n = true ,
597   columns-width .code:n =
598     \tl_if_eq:nnTF { #1 } { auto }
599     { \bool_set_true:N \l_@@_auto_columns_width_bool }
600     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
601   columns-width .value_required:n = true ,
602   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.

```

603   \legacy_if:nF { measuring@ }
604   {

```

```

605     \str_set:Nn \l_tmpa_str { #1 }
606     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
607     { \@@_error:nn { Duplicate-name } { #1 } }
608     { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
609     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
610   } ,
611   name .value_required:n = true ,
612   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
613   code-after .value_required:n = true ,
614   colortbl-like .code:n =
615     \bool_set_true:N \l_@@_colortbl_like_bool
616     \bool_set_true:N \l_@@_code_before_bool ,
617   colortbl-like .value_forbidden:n = true
618 }
619 \keys_define:nn { NiceMatrix / notes }
620 {
621   para .bool_set:N = \l_@@_notes_para_bool ,
622   para .default:n = true ,
623   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
624   code-before .value_required:n = true ,
625   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
626   code-after .value_required:n = true ,
627   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
628   bottomrule .default:n = true ,
629   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
630   style .value_required:n = true ,
631   label-in-tabular .code:n =
632     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
633   label-in-tabular .value_required:n = true ,
634   label-in-list .code:n =
635     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
636   label-in-list .value_required:n = true ,
637   enumitem-keys .code:n =
638     {
639       \bool_if:NTF \c_@@_in_preamble_bool
640       {
641         \AtBeginDocument
642         {
643           \bool_if:NT \c_@@_enumitem_loaded_bool
644           { \setlist* [ tabularnotes ] { #1 } }
645         }
646       }
647       {
648         \bool_if:NT \c_@@_enumitem_loaded_bool
649         { \setlist* [ tabularnotes ] { #1 } }
650       }
651     } ,
652   enumitem-keys .value_required:n = true ,
653   enumitem-keys-para .code:n =
654     {
655       \bool_if:NTF \c_@@_in_preamble_bool
656       {
657         \AtBeginDocument
658         {
659           \bool_if:NT \c_@@_enumitem_loaded_bool
660           { \setlist* [ tabularnotes* ] { #1 } }
661         }
662       }
663       {
664         \bool_if:NT \c_@@_enumitem_loaded_bool
665         { \setlist* [ tabularnotes* ] { #1 } }
666       }
667     } ,

```

```

668   enumitem-keys-para .value_required:n = true ,
669   unknown .code:n = \@@_error:n { Unknown-key-for-notes }
670 }

```

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

```

671 \keys_define:nn { NiceMatrix }
672 {
673   NiceMatrixOptions .inherit:n =
674     { NiceMatrix / Global } ,
675   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
676   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
677   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
678   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
679   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
680   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
681   NiceMatrix .inherit:n =
682     {
683       NiceMatrix / Global ,
684       NiceMatrix / Env ,
685     } ,
686   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
687   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
688   NiceTabular .inherit:n =
689     {
690       NiceMatrix / Global ,
691       NiceMatrix / Env
692     } ,
693   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
694   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
695   NiceArray .inherit:n =
696     {
697       NiceMatrix / Global ,
698       NiceMatrix / Env ,
699     } ,
700   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
701   NiceArray / rules .inherit:n = NiceMatrix / rules ,
702   pNiceArray .inherit:n =
703     {
704       NiceMatrix / Global ,
705       NiceMatrix / Env ,
706     } ,
707   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
708   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
709 }

```

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

```

710 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
711 {
712   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
713   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
714   delimiters / color .value_required:n = true ,
715   delimiters-color .tl_set:N = \l_@@_delimiters_color_tl ,
716   delimiters-color .value_required:n = true ,
717   last-col .code:n = \tl_if_empty:nF { #1 }
718     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
719     \int_zero:N \l_@@_last_col_int ,
720   small .bool_set:N = \l_@@_small_bool ,
721   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.

```

722   renew-matrix .code:n = \@@_renew_matrix: ,
723   renew-matrix .value_forbidden:n = true ,

```

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

```

724   transparent .code:n =
725   {
726     \@@_renew_matrix:
727     \bool_set_true:N \l_@@_renew_dots_bool
728     \@@_error:n { Key-transparent }
729   } ,
730   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}`.

```

731   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.

```

732   columns-width .code:n =
733   \tl_if_eq:nnTF { #1 } { auto }
734   { \@@_error:n { Option-auto-for-columns-width } }
735   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

736   allow-duplicate-names .code:n =
737   \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
738   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`).

```

739   letter-for-dotted-lines .code:n =
740   {
741     \tl_if_single_token:nTF { #1 }
742     { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
743     { \@@_error:n { One-letter-allowed } }
744   } ,
745   letter-for-dotted-lines .value_required:n = true ,
746   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
747   notes .value_required:n = true ,
748   sub-matrix .code:n =
749   \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
750   sub-matrix .value_required:n = true ,
751   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
752 }

753 \str_new:N \l_@@_letter_for_dotted_lines_str
754 \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.

```

755 \NewDocumentCommand \NiceMatrixOptions { m }
756 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

757 \keys_define:nn { NiceMatrix / NiceMatrix }
758 {
759   last-col .code:n = \tl_if_empty:nTF {#1}
760     {
761       \bool_set_true:N \l_@@_last_col_without_value_bool
762       \int_set:Nn \l_@@_last_col_int { -1 }
763     }
764     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
765   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
766   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
767   small .bool_set:N = \l_@@_small_bool ,
768   small .value_forbidden:n = true ,
769   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
770   delimiters / color .value_required:n = true ,
771   delimiters-color .tl_set:N = \l_@@_delimiters_color_tl ,
772   delimiters-color .value_required:n = true ,
773   unknown .code:n = \@@_error:n { Unknown-option-for-NiceMatrix }
774 }

```

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

```

775 \keys_define:nn { NiceMatrix / NiceArray }
776 {

```

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.

```

777   small .bool_set:N = \l_@@_small_bool ,
778   small .value_forbidden:n = true ,
779   last-col .code:n = \tl_if_empty:nF { #1 }
780     { \@@_error:n { last-col-non-empty-for-NiceArray } }
781     { \int_zero:N \l_@@_last_col_int ,
782   notes / para .bool_set:N = \l_@@_notes_para_bool ,
783   notes / para .default:n = true ,
784   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
785   notes / bottomrule .default:n = true ,
786   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
787   tabularnote .value_required:n = true ,
788   delimiters-color .tl_set:N = \l_@@_delimiters_color_tl ,
789   delimiters-color .value_required:n = true ,
790   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
791   delimiters / color .value_required:n = true ,
792   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
793   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
794   unknown .code:n = \@@_error:n { Unknown-option-for-NiceArray }
795 }
796 \keys_define:nn { NiceMatrix / pNiceArray }
797 {
798   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
799   last-col .code:n = \tl_if_empty:nF {#1}
800     { \@@_error:n { last-col-non-empty-for-NiceArray } }
801     { \int_zero:N \l_@@_last_col_int ,
802   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
803   small .bool_set:N = \l_@@_small_bool ,
804   small .value_forbidden:n = true ,
805   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
806   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
807   unknown .code:n = \@@_error:n { Unknown-option-for-NiceMatrix }
808 }

```

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

```

809 \keys_define:nn { NiceMatrix / NiceTabular }
810 {
811   notes / para .bool_set:N = \l_@@_notes_para_bool ,
812   notes / para .default:n = true ,
813   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
814   notes / bottomrule .default:n = true ,
815   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
816   tabularnote .value_required:n = true ,
817   last-col .code:n = \tl_if_empty:nF {#1}
818     { \@@_error:n { last-col~non-empty~for~NiceArray } }
819     \int_zero:N \l_@@_last_col_int ,
820   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
821   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
822   unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular }
823 }

```

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}).

```

824 \cs_new_protected:Npn \@@_Cell:
825 {

```

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`).

```

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

```

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

```

827   \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.

```

828   \int_compare:nNnT \c@jCol = 1
829     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
830   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }

```

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

```

831   \hbox_set:Nw \l_@@_cell_box
832   \bool_if:NF \l_@@_NiceTabular_bool
833   {
834     \c_math_toggle_token
835     \bool_if:NT \l_@@_small_bool \scriptstyle
836   }

```

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.

```

837   \int_compare:nNnTF \c@iRow = 0
838   {
839     \int_compare:nNnT \c@jCol > 0
840     {
841       \l_@@_code_for_first_row_tl
842       \xglobal \colorlet { nicematrix-first-row } { . }
843     }
844   }

```

```

845 {
846   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
847   {
848     \l_@@_code_for_last_row_tl
849     \xglobal \colorlet { nicematrix-last-row } { . }
850   }
851 }
852 }

```

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.

```

853 \cs_new_protected:Npn \@@_begin_of_row:
854 {
855   \int_gincr:N \c@iRow
856   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
857   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
858   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
859   \pgfpicture
860   \pgfrememberpicturepositiononpagetrue
861   \pgfcoordinate
862   { \@@_env: - row - \int_use:N \c@iRow - base }
863   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
864   \str_if_empty:NF \l_@@_name_str
865   {
866     \pgfnodealias
867     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
868     { \@@_env: - row - \int_use:N \c@iRow - base }
869   }
870   \endpgfpicture
871 }

```

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.

```

872 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
873 {
874   \int_compare:nNnTF \c@iRow = 0
875   {
876     \dim_gset:Nn \g_@@_dp_row_zero_dim
877     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
878     \dim_gset:Nn \g_@@_ht_row_zero_dim
879     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
880   }
881   {
882     \int_compare:nNnT \c@iRow = 1
883     {
884       \dim_gset:Nn \g_@@_ht_row_one_dim
885       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
886     }
887   }
888 }
889 \cs_new_protected:Npn \@@_rotate_cell_box:
890 {
891   \box_rotate:Nn \l_@@_cell_box { 90 }
892   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
893   {
894     \vbox_set_top:Nn \l_@@_cell_box
895     {
896       \vbox_to_zero:n { }
897       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }

```

```

898         \box_use:N \l_@@_cell_box
899     }
900 }
901 \bool_gset_false:N \g_@@_rotate_bool
902 }
903 \cs_new_protected:Npn \@@_adjust_size_box:
904 {
905     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
906     {
907         \box_set_wd:Nn \l_@@_cell_box
908         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
909         \dim_gzero:N \g_@@_blocks_wd_dim
910     }
911     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
912     {
913         \box_set_dp:Nn \l_@@_cell_box
914         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
915         \dim_gzero:N \g_@@_blocks_dp_dim
916     }
917     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
918     {
919         \box_set_ht:Nn \l_@@_cell_box
920         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
921         \dim_gzero:N \g_@@_blocks_ht_dim
922     }
923 }
924 \cs_new_protected:Npn \@@_end_Cell:
925 {
926     \@@_math_toggle_token:
927     \hbox_set_end:
928     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
929     \@@_adjust_size_box:
930     \box_set_ht:Nn \l_@@_cell_box
931     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
932     \box_set_dp:Nn \l_@@_cell_box
933     { \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”).

```

934     \dim_gset:Nn \g_@@_max_cell_width_dim
935     { \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”.

```

936     \@@_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.


```

937 \bool_if:NTF \g_@@_empty_cell_bool
938 { \box_use_drop:N \l_@@_cell_box }
939 {
940   \bool_lazy_or:nnTF
941     \g_@@_not_empty_cell_bool
942     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
943     \@@_node_for_the_cell:
944     { \box_use_drop:N \l_@@_cell_box }
945   }
946   \bool_gset_false:N \g_@@_empty_cell_bool
947   \bool_gset_false:N \g_@@_not_empty_cell_bool
948 }

```

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

```

949 \cs_new_protected:Npn \@@_node_for_the_cell:
950 {
951   \pgfpicture
952   \pgfsetbaseline \c_zero_dim
953   \pgfrememberpicturepositiononpagetrue
954   \pgfset
955   {
956     inner~sep = \c_zero_dim ,
957     minimum~width = \c_zero_dim
958   }
959   \pgfnode
960   { rectangle }
961   { base }
962   { \box_use_drop:N \l_@@_cell_box }
963   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
964   { }
965   \str_if_empty:NF \l_@@_name_str
966   {
967     \pgfnodealias
968     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
969     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
970   }
971   \endpgfpicture
972 }

```

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.

```

973 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
974 {
975   \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
976   { g_@@_ #2 _ lines _ tl }

```

```

977 {
978   \use:c { @@ _ draw _ #2 : nnn }
979   { \int_use:N \c@iRow }
980   { \int_use:N \c@jCol }
981   { \exp_not:n { #3 } }
982 }
983 }

```

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`.

```

984 \cs_new_protected:Npn \@@_revtex_array:
985 {
986   \cs_set_eq:NN \@acoll \@arrayacol
987   \cs_set_eq:NN \@acolr \@arrayacol
988   \cs_set_eq:NN \@acol \@arrayacol
989   \cs_set_nopar:Npn \@halignto { }
990   \@array@array
991 }
992 \cs_new_protected:Npn \@@_array:
993 {
994   \bool_if:NTF \c_@@_revtex_bool
995     \@@_revtex_array:
996   {
997     \bool_if:NTF \l_@@_NiceTabular_bool
998       { \dim_set_eq:NN \col@sep \tabcolsep }
999       { \dim_set_eq:NN \col@sep \arraycolsep }
1000     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1001       { \cs_set_nopar:Npn \@halignto { } }
1002       { \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`.

```

1003   \@tabarray
1004 }

```

`\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.

```

1005   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1006 }

```

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.

```

1007 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1008 \cs_new_protected:Npn \@@_create_row_node:
1009 {

```

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

```

1010   \hbox
1011   {
1012     \bool_if:NT \l_@@_code_before_bool
1013     {
1014       \vtop
1015       {
1016         \skip_vertical:N 0.5\arrayrulewidth
1017         \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
1018         \skip_vertical:N -0.5\arrayrulewidth
1019       }
1020     }
1021     \pgfpicture
1022     \pgfrememberpicturepositiononpagetrue

```

```

1023 \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
1024 { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1025 \str_if_empty:NF \l_@@_name_str
1026 {
1027 \pgfnodealias
1028 { \l_@@_name_str - row - \int_use:N \c@iRow }
1029 { \@@_env: - row - \int_use:N \c@iRow }
1030 }
1031 \endpgfpicture
1032 }
1033 }

```

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

```

1034 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1035 \cs_new_protected:Npn \@@_everycr_i:
1036 {
1037 \int_gzero:N \c@jCol
1038 \bool_gset_false:N \g_@@_after_col_zero_bool
1039 \bool_if:NF \g_@@_row_of_col_done_bool
1040 {
1041 \@@_create_row_node:

```

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

```

1042 \tl_if_empty:NF \l_@@_hlines_clist
1043 {
1044 \tl_if_eq:NnF \l_@@_hlines_clist { all }
1045 {
1046 \exp_args:NNx
1047 \clist_if_in:NnT
1048 \l_@@_hlines_clist
1049 { \@@_succ:n \c@iRow }
1050 }
1051 {

```

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.

```

1052 \int_compare:nNnT \c@iRow > { -1 }
1053 {
1054 \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@`.

```

1055 { \hrule height \arrayrulewidth width \c_zero_dim }
1056 }
1057 }
1058 }
1059 }
1060 }

```

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

```

1061 \cs_set_protected:Npn \@@_newcolumntype #1
1062 {
1063 \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1064 \peek_meaning:NTF [
1065 { \newcol@ #1 }
1066 { \newcol@ #1 [ 0 ] }
1067 }

```

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

```

1068 \cs_set_protected:Npn \@@_renew_dots:
1069 {
1070   \cs_set_eq:NN \ldots \@@_Ldots
1071   \cs_set_eq:NN \cdots \@@_Cdots
1072   \cs_set_eq:NN \vdots \@@_Vdots
1073   \cs_set_eq:NN \ddots \@@_Ddots
1074   \cs_set_eq:NN \iddots \@@_Iddots
1075   \cs_set_eq:NN \dots \@@_Ldots
1076   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1077 }

```

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

```

1078 \cs_new_protected:Npn \@@_colortbl_like:
1079 {
1080   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1081   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1082   \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1083 }

```

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

```

1084 \cs_new_protected:Npn \@@_pre_array_ii:
1085 {

```

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 ⁴⁶.

```

1086   \bool_if:NT \c_@@_booktabs_loaded_bool
1087     { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1088   \box_clear_new:N \l_@@_cell_box
1089   \cs_if_exist:NT \theiRow
1090     { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1091   \int_gzero_new:N \c@iRow
1092   \cs_if_exist:NT \thejCol
1093     { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1094   \int_gzero_new:N \c@jCol
1095   \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}`).

```

1096   \bool_if:NT \l_@@_small_bool
1097     {
1098       \cs_set_nopar:Npn \arraystretch { 0.47 }
1099       \dim_set:Nn \arraycolsep { 1.45 pt }
1100     }

```

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`.

```

1101   \cs_set_nopar:Npn \ialign
1102     {
1103       \bool_if:NTF \c_@@_colortbl_loaded_bool
1104         {

```

⁴⁶cf. `\nicematrix@redefine@check@rerun`

```

1105         \CT@everycr
1106         {
1107             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1108             \@@_everycr:
1109         }
1110     }
1111     { \everycr { \@@_everycr: } }
1112     \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`⁴⁷ 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`.

```

1113     \dim_gzero_new:N \g_@@_dp_row_zero_dim
1114     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1115     \dim_gzero_new:N \g_@@_ht_row_zero_dim
1116     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1117     \dim_gzero_new:N \g_@@_ht_row_one_dim
1118     \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1119     \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1120     \dim_gzero_new:N \g_@@_ht_last_row_dim
1121     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1122     \dim_gzero_new:N \g_@@_dp_last_row_dim
1123     \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`.

```

1124     \cs_set_eq:NN \ialign \@@_old_ialign:
1125     \halign
1126 }

```

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).

```

1127     \cs_set_eq:NN \@@_old_ldots \ldots
1128     \cs_set_eq:NN \@@_old_cdots \cdots
1129     \cs_set_eq:NN \@@_old_vdots \vdots
1130     \cs_set_eq:NN \@@_old_ddots \ddots
1131     \cs_set_eq:NN \@@_old_iddots \iddots
1132     \bool_if:NTF \l_@@_standard_cline_bool
1133     { \cs_set_eq:NN \cline \@@_standard_cline }
1134     { \cs_set_eq:NN \cline \@@_cline }
1135     \cs_set_eq:NN \Ldots \@@_Ldots
1136     \cs_set_eq:NN \Cdots \@@_Cdots
1137     \cs_set_eq:NN \Vdots \@@_Vdots
1138     \cs_set_eq:NN \Ddots \@@_Ddots
1139     \cs_set_eq:NN \Iddots \@@_Iddots
1140     \cs_set_eq:NN \hdottedline \@@_hdottedline:
1141     \cs_set_eq:NN \Hline \@@_Hline:
1142     \cs_set_eq:NN \Hspace \@@_Hspace:
1143     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1144     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1145     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1146     \cs_set_eq:NN \Block \@@_Block:
1147     \cs_set_eq:NN \rotate \@@_rotate:
1148     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1149     \cs_set_eq:NN \dotfill \@@_old_dotfill:
1150     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1151     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1152     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:

```

⁴⁷The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1153 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1154 \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).

```

1155 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1156 \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).

```

1157 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1158 \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.

```

1159 \int_gzero_new:N \g_@@_col_total_int
1160 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1161 \@@_renew_NC@rewrite@S:
1162 \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.

```

1163 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1164 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1165 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1166 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1167 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1168 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1169 \tl_gclear_new:N \g_nicematrix_code_before_tl
1170 }

```

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

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

```

1171 \cs_new_protected:Npn \@@_pre_array:
1172 {
1173   \seq_gclear:N \g_@@_submatrix_seq
1174   \bool_if:NT \l_@@_code_before_bool
1175   {
1176     \seq_if_exist:cT { @@_size_ \int_use:N \g_@@_env_int _ seq }
1177     {

```

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).

```

1178 \int_zero_new:N \c@iRow
1179 \int_set:Nn \c@iRow
1180 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
1181 \int_zero_new:N \c@jCol
1182 \int_set:Nn \c@jCol
1183 { \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.

```

1184 \int_compare:nNnF \l_@@_last_row_int = { -2 }
1185 { \int_decr:N \c@iRow }
1186 \int_compare:nNnF \l_@@_last_col_int = { -2 }
1187 { \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.

```

1188 \pgfsys@markposition { \@@_env: - position }
1189 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1190 \pgfpicture

```

First, the creation of the row nodes.

```

1191 \int_step_inline:nnn
1192 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 1 }
1193 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 + 1 }
1194 {
1195   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1196   \pgfcoordinate { \@@_env: - row - ##1 }
1197   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1198 }

```

Now, the creation of the col nodes.

```

1199 \int_step_inline:nnn
1200 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
1201 { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 + 1 }
1202 {
1203   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1204   \pgfcoordinate { \@@_env: - col - ##1 }
1205   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1206 }
1207 \endpgfpicture

```

Now, you create the diagonal nodes by using the row nodes and the col nodes.

```

1208 \@@_create_diag_nodes:

```

Now, yet other settings before the execution of the code-before.

```

1209 \group_begin:
1210 \bool_if:NT \c_@@_tikz_loaded_bool
1211 {
1212   \tikzset
1213   {
1214     every~picture / .style =
1215     { overlay , name~prefix = \@@_env: - }
1216   }
1217 }
1218 \cs_set_eq:NN \cellcolor \@@_cellcolor
1219 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1220 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1221 \cs_set_eq:NN \rowcolor \@@_rowcolor
1222 \cs_set_eq:NN \rowcolors \@@_rowcolors
1223 \cs_set_eq:NN \columncolor \@@_columncolor
1224 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1225 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before

```

We compose the code-before in math mode in order to nullify the spaces put by the user between instructions in the code-before.

```

1226 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1227 \seq_clear_new:N \l_@@_colors_seq

```

Here is the `\CodeBefore`. As of now, the keys that may be provided to the keyword `\CodeBefore` are the same as keys that may be provided to `\CodeAfter`, hence the `\@@_CodeAfter_keys:`.

```

1228 \exp_last_unbraced:NV \@@_CodeAfter_keys: \l_@@_code_before_tl

```

Now, all the cells which are specified to be colored by instructions in the `\CodeBefore` will actually be colored. It's a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

```

1229         \@@_actually_color:
1230         \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1231     \group_end:
1232 }
1233 }

```

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).

```

1234 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1235 {
1236     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1237     {
1238         \dim_gset:Nn \g_@@_ht_last_row_dim
1239         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1240         \dim_gset:Nn \g_@@_dp_last_row_dim
1241         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1242     }
1243 }
1244 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1245 {
1246     \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.

```

1247 \str_if_empty:NTF \l_@@_name_str
1248 {
1249     \cs_if_exist:cT { @@_last_row_ \int_use:N \g_@@_env_int }
1250     {
1251         \int_set:Nn \l_@@_last_row_int
1252         { \use:c { @@_last_row_ \int_use:N \g_@@_env_int } }
1253     }
1254 }
1255 {
1256     \cs_if_exist:cT { @@_last_row_ \l_@@_name_str }
1257     {
1258         \int_set:Nn \l_@@_last_row_int
1259         { \use:c { @@_last_row_ \l_@@_name_str } }
1260     }
1261 }
1262 }

```

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).

```

1263 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1264 {
1265     \str_if_empty:NTF \l_@@_name_str
1266     {
1267         \cs_if_exist:cT { @@_last_col_ \int_use:N \g_@@_env_int }
1268         {
1269             \int_set:Nn \l_@@_last_col_int
1270             { \use:c { @@_last_col_ \int_use:N \g_@@_env_int } }
1271         }
1272     }
1273     {
1274         \cs_if_exist:cT { @@_last_col_ \l_@@_name_str }
1275         {
1276             \int_set:Nn \l_@@_last_col_int
1277             { \use:c { @@_last_col_ \l_@@_name_str } }
1278         }
1279     }

```



```
1280 }
```

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

```
1281 \@@_pre_array_ii:
```

We compute the width of both delimiters.

```
1282 \dim_zero_new:N \l_@@_left_delim_dim
1283 \dim_zero_new:N \l_@@_right_delim_dim
1284 \bool_if:NTF \l_@@_NiceArray_bool
1285 {
1286   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1287   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1288 }
1289 {
```

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

```
1290 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \l_@@_left_delim_tl $ }
1291 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1292 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \l_@@_right_delim_tl $ }
1293 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1294 }
```

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

```
1295 \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).

```
1296 \bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:
```

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

```
1297 \@@_construct_preamble:
```

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

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

```
1298 \hbox_set:Nw \l_@@_the_array_box
1299 \skip_horizontal:N \l_@@_left_margin_dim
1300 \skip_horizontal:N \l_@@_extra_left_margin_dim
1301 \c_math_toggle_token
1302 \bool_if:NTF \l_@@_light_syntax_bool
1303 { \use:c { @@-light-syntax } }
1304 { \use:c { @@-normal-syntax } }
1305 }
```

The following command `\@@_pre_array_i:w` will be used when the keyword `\CodeBefore` is present at the beginning of the environment.

```
1306 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1307 {
1308   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1309   \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...

```
1310 \@@_pre_array:
1311 }
```

The environment {NiceArrayWithDelims}

```

1312 \NewDocumentEnvironment { NiceArrayWithDelims }
1313 { m m 0 { } m ! 0 { } t \CodeBefore }
1314 {
1315   \@@_provide_pgfsyspdfmark:
1316   \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.

```

1317   \bgroup

1318   \tl_set:Nn \l_@@_left_delim_tl { #1 }
1319   \tl_set:Nn \l_@@_right_delim_tl { #2 }
1320   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1321   \int_gzero:N \g_@@_block_box_int
1322   \dim_zero:N \g_@@_width_last_col_dim
1323   \dim_zero:N \g_@@_width_first_col_dim
1324   \bool_gset_false:N \g_@@_row_of_col_done_bool
1325   \str_if_empty:NT \g_@@_name_env_str
1326   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1327   \@@_adapt_S_column:
1328   \bool_if:NTF \l_@@_NiceTabular_bool
1329     \mode_leave_vertical:
1330     \@@_test_if_math_mode:
1331   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1332   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁴⁸. 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.

```

1333   \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.

```

1334   \cs_if_exist:NT \tikz@library@external@loaded
1335   {
1336     \tikzexternaldisable
1337     \cs_if_exist:NT \ifstandalone
1338     { \tikzset { external / optimize = false } }
1339   }

```

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

```

1340   \int_gincr:N \g_@@_env_int
1341   \bool_if:NF \l_@@_block_auto_columns_width_bool
1342   { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1343   \seq_gclear:N \g_@@_blocks_seq
1344   \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`.

```

1345   \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1346   \seq_gclear:N \g_@@_pos_of_xdots_seq

1347   \tl_if_exist:cT { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
1348   {
1349     \bool_set_true:N \l_@@_code_before_bool

```

⁴⁸e.g. `\color[rgb]{0.5,0.5,0}`

```

1350 \exp_args:NNv \tl_put_right:Nn \l_@@_code_before_tl
1351 { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
1352 }

```

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.

```

1353 \bool_if:NTF \l_@@_NiceArray_bool
1354 { \keys_set:nn { NiceMatrix / NiceArray } }
1355 { \keys_set:nn { NiceMatrix / pNiceArray } }
1356 { #3 , #5 }

1357 \tl_if_empty:NF \l_@@_rules_color_tl
1358 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
1359 % \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:.

```

1360 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
1361 }
1362 {
1363   \bool_if:NTF \l_@@_light_syntax_bool
1364   { \use:c { end @@-light-syntax } }
1365   { \use:c { end @@-normal-syntax } }
1366   \c_math_toggle_token
1367   \skip_horizontal:N \l_@@_right_margin_dim
1368   \skip_horizontal:N \l_@@_extra_right_margin_dim
1369   \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).

```

1370 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1371 {
1372   \bool_if:NF \l_@@_last_row_without_value_bool
1373   {
1374     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1375     {
1376       \@@_error:n { Wrong~last~row }
1377       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1378     }
1379   }
1380 }

```

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”.⁴⁹

```

1381 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1382 \bool_if:nTF \g_@@_last_col_found_bool
1383 { \int_gdecr:N \c@jCol }
1384 {
1385   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1386   { \@@_error:n { last~col~not~used } }
1387 }

```

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

```

1388 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1389 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

⁴⁹We remind that the potential “first column” (exterior) has the number 0.

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. 100).

```

1390   \int_compare:nNnT \l_@@_first_col_int = 0
1391   {
1392     \skip_horizontal:N \col@sep
1393     \skip_horizontal:N \g_@@_width_first_col_dim
1394   }

```

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.

```

1395   \bool_if:NTF \l_@@_NiceArray_bool
1396   {
1397     \str_case:VnF \l_@@_baseline_tl
1398     {
1399       b \@@_use_arraybox_with_notes_b:
1400       c \@@_use_arraybox_with_notes_c:
1401     }
1402     \@@_use_arraybox_with_notes:
1403   }

```

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).

```

1404   {
1405     \int_compare:nNnTF \l_@@_first_row_int = 0
1406     {
1407       \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1408       \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1409     }
1410     { \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”.⁵⁰

```

1411   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1412   {
1413     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1414     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1415   }
1416   { \dim_zero:N \l_tmpb_dim }
1417   \hbox_set:Nn \l_tmpa_box
1418   {
1419     \c_math_toggle_token
1420     \tl_if_empty:NF \l_@@_delimiters_color_tl
1421     { \color { \l_@@_delimiters_color_tl } }
1422     \exp_after:wN \left \l_@@_left_delim_tl
1423     \vcenter
1424     {

```

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.

```

1425       \skip_vertical:N -\l_tmpa_dim
1426       \hbox
1427       {
1428         \bool_if:NTF \l_@@_NiceTabular_bool
1429         { \skip_horizontal:N -\tabcolsep }
1430         { \skip_horizontal:N -\arraycolsep }
1431         \@@_use_arraybox_with_notes_c:
1432         \bool_if:NTF \l_@@_NiceTabular_bool
1433         { \skip_horizontal:N -\tabcolsep }
1434         { \skip_horizontal:N -\arraycolsep }

```

⁵⁰A 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).

```

1435         }
We take into account the “last row” (we have previously computed its total height in \l_tmpb_dim).
1436         \skip_vertical:N -\l_tmpb_dim
1437     }

```

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.

```

1438         \tl_if_empty:NF \l_@@_delimiters_color_tl
1439         { \color { \l_@@_delimiters_color_tl } }
1440         \exp_after:wN \right \l_@@_right_delim_tl
1441         \c_math_toggle_token
1442     }

```

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.

```

1443         \bool_if:NTF \l_@@_delimiters_max_width_bool
1444         { \@@_put_box_in_flow_bis:nn { #1 } { #2 } }
1445         \@@_put_box_in_flow:
1446     }

```

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. 101).

```

1447         \bool_if:NT \g_@@_last_col_found_bool
1448         {
1449             \skip_horizontal:N \g_@@_width_last_col_dim
1450             \skip_horizontal:N \col@sep
1451         }
1452         \bool_if:NF \l_@@_Matrix_bool
1453         {
1454             \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1455             { \@@_error:n { columns-not-used } }
1456         }
1457         \group_begin:
1458         \globaldefs = 1
1459         \@@_msg_redirect_name:nn { columns-not-used } { error }
1460         \group_end:
1461         \@@_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.

```

1462         \egroup
1463         \bool_if:NT \c_@@_footnote_bool \endsavenotes
1464     }

```

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 use is in `\g_@@_preamble_tl` and the modified version will be stored in `\g_@@_preamble_tl` also.

```

1465 \cs_new_protected:Npn \@@_construct_preamble:
1466 {

```

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.

```
1467 \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).

```
1468 \bool_if:NF \l_@@_Matrix_bool
1469 {
1470   \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1471   \@@_newcolumnntype 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`).

```
1472 \exp_args:NV \@temptokena \g_@@_preamble_tl
```

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

```
1473 \@tempswatrue
```

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

```
1474 \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
```

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

```
1475 \int_gzero_new:N \c@jCol
1476 \tl_gclear:N \g_@@_preamble_tl
1477 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1478 {
1479   \tl_gset:Nn \g_@@_preamble_tl
1480     { ! { \skip_horizontal:N \arrayrulewidth } }
1481 }
1482 {
1483   \clist_if_in:NnT \l_@@_vlines_clist 1
1484   {
1485     \tl_gset:Nn \g_@@_preamble_tl
1486       { ! { \skip_horizontal:N \arrayrulewidth } }
1487   }
1488 }
```

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`).

```
1489 \seq_clear:N \g_@@_cols_vlism_seq
```

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

```
1490 \int_zero:N \l_tmpa_int
```

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

```
1491 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1492 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1493 }
```

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

```
1494 \bool_if:NT \l_@@_colortbl_like_bool
1495 {
1496   \regex_replace_all:NnN
1497     \c_@@_columncolor_regex
1498     { \c { @@_columncolor_preamble } }
1499     \g_@@_preamble_tl
1500 }
```

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

```

1501 \int_compare:nNnTF \l_@@_first_col_int = 0
1502 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1503 {
1504   \bool_lazy_all:nT
1505   {
1506     \l_@@_NiceArray_bool
1507     { \bool_not_p:n \l_@@_NiceTabular_bool }
1508     { \tl_if_empty_p:N \l_@@_vlines_clist }
1509     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1510   }
1511   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1512 }
1513 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1514 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1515 {
1516   \bool_lazy_all:nT
1517   {
1518     \l_@@_NiceArray_bool
1519     { \bool_not_p:n \l_@@_NiceTabular_bool }
1520     { \tl_if_empty_p:N \l_@@_vlines_clist }
1521     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1522   }
1523   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1524 }

```

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`).

```

1525 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1526 {
1527   \tl_gput_right:Nn \g_@@_preamble_tl
1528   { > { \@@_error_too_much_cols: } 1 }
1529 }

```

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

```

1530 \group_end:
1531 }

1532 \cs_new_protected:Npn \@@_patch_preamble:n #1
1533 {
1534   \str_case:nnF { #1 }
1535   {
1536     c { \@@_patch_preamble_i:n #1 }
1537     l { \@@_patch_preamble_i:n #1 }
1538     r { \@@_patch_preamble_i:n #1 }
1539     > { \@@_patch_preamble_ii:nn #1 }
1540     ! { \@@_patch_preamble_ii:nn #1 }
1541     @ { \@@_patch_preamble_ii:nn #1 }
1542     | { \@@_patch_preamble_iii:n #1 }
1543     p { \@@_patch_preamble_iv:nnn t #1 }
1544     m { \@@_patch_preamble_iv:nnn c #1 }
1545     b { \@@_patch_preamble_iv:nnn b #1 }
1546     \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
1547     \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1548     \@@_true_c: { \@@_patch_preamble_vi:n #1 }
1549     ( { \@@_patch_preamble_vii:n #1 }
1550     [ { \@@_patch_preamble_vii:n #1 }
1551     \{ { \@@_patch_preamble_vii:n #1 }
1552     ) { \@@_patch_preamble_viii:n #1 }
1553     ] { \@@_patch_preamble_viii:n #1 }
1554     \} { \@@_patch_preamble_viii:n #1 }

```

```

1555     C { \@@_error:nn { old~column~type } #1 }
1556     L { \@@_error:nn { old~column~type } #1 }
1557     R { \@@_error:nn { old~column~type } #1 }
1558     \q_stop { }
1559   }
1560   {
1561     \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1562     { \@@_patch_preamble_xi:n #1 }
1563     {
1564       \str_if_eq:VnTF \l_@@_letter_vlism_tl { #1 }
1565       {
1566         \seq_gput_right:Nx \g_@@_cols_vlism_seq
1567         { \int_eval:n { \c@jCol + 1 } }
1568         \tl_gput_right:Nx \g_@@_preamble_tl
1569         { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1570         \@@_patch_preamble:n
1571       }
1572       {
1573         \bool_lazy_and:nnTF
1574         { \str_if_eq_p:nn { : } { #1 } }
1575         \c_@@_arydshln_loaded_bool
1576         {
1577           \tl_gput_right:Nn \g_@@_preamble_tl { : }
1578           \@@_patch_preamble:n
1579         }
1580         { \@@_fatal:nn { unknown~column~type } { #1 } }
1581       }
1582     }
1583   }
1584 }

```

For c, l and r

```

1585 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1586 {
1587   \tl_gput_right:Nn \g_@@_preamble_tl
1588   {
1589     > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } }
1590     #1
1591     < \@@_end_Cell:
1592   }

```

We increment the counter of columns and then we test for the presence of a <.

```

1593   \int_gincr:N \c@jCol
1594   \@@_patch_preamble_x:n
1595 }

```

For >, ! and @

```

1596 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1597 {
1598   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1599   \@@_patch_preamble:n
1600 }

```

For |

```

1601 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1602 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1603   \int_incr:N \l_tmpa_int
1604   \@@_patch_preamble_iii_i:n
1605 }

```



```

1606 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1607 {
1608   \str_if_eq:nnTF { #1 } |
1609   { \@@_patch_preamble_iii:n | }
1610   {
1611     \tl_gput_right:Nx \g_@@_preamble_tl
1612     {
1613       \exp_not:N !
1614       {
1615         \skip_horizontal:n
1616         {
1617           \dim_eval:n
1618           {
1619             \arrayrulewidth * \l_tmpa_int
1620             + \doublerulesep * ( \l_tmpa_int - 1)
1621           }
1622         }
1623       }
1624     }
1625     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1626     { \@@_vline:nn { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } }
1627     \int_zero:N \l_tmpa_int
1628     \@@_patch_preamble:n #1
1629   }
1630 }

```

For p, m and b

```

1631 \cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
1632 {
1633   \tl_gput_right:Nn \g_@@_preamble_tl
1634   {
1635     > {
1636       \@@_Cell:
1637       \begin { minipage } [ #1 ] { #3 }
1638       \mode_leave_vertical:
1639       \arraybackslash
1640       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt % v. 5.11
1641     }
1642     c
1643     < {
1644       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt % v. 5.11
1645       \end { minipage }
1646       \@@_end_Cell:
1647     }
1648   }

```

We increment the counter of columns, and then we test for the presence of a <.

```

1649   \int_gincr:N \c@jCol
1650   \@@_patch_preamble_x:n
1651 }

```

For w and W

```

1652 \cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
1653 {
1654   \tl_gput_right:Nn \g_@@_preamble_tl
1655   {
1656     > {
1657       \hbox_set:Nw \l_@@_cell_box
1658       \@@_Cell:
1659       \tl_set:Nn \l_@@_cell_type_tl { #3 }
1660     }
1661     c
1662     < {
1663       \@@_end_Cell:

```

```

1664         #1
1665         \hbox_set_end:
1666         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1667         \@@_adjust_size_box:
1668         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
1669     }
1670 }

```

We increment the counter of columns and then we test for the presence of a <.

```

1671     \int_gincr:N \c@jCol
1672     \@@_patch_preamble_x:n
1673 }

```

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

```

1674 \cs_new_protected:Npn \@@_patch_preamble_vi:n #1
1675 {
1676     \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

We increment the counter of columns and then we test for the presence of a <.

```

1677     \int_gincr:N \c@jCol
1678     \@@_patch_preamble_x:n
1679 }

```

For (, [and \{.

```

1680 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
1681 {
1682     \bool_if:NT \l_@@_small_bool
1683     { \@@_fatal:n { Delimiter-with-small } }

```

If we are before the column 1, we reserve space for the left delimiter.

```

1684     \int_compare:nNnT \c@jCol = \c_zero_int
1685     { \tl_gput_right:Nx \g_@@_preamble_tl { ! { \enskip } } }
1686     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1687     { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
1688     \@@_patch_preamble:n
1689 }

```

For),] and \}.

```

1690 \cs_new_protected:Npn \@@_patch_preamble_viii:n #1
1691 {
1692     \bool_if:NT \l_@@_small_bool
1693     { \@@_fatal:n { Delimiter-with-small } }
1694     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1695     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }

```

After this closing delimiter, we have to test whether there is a opening delimiter (we consider that there can't be a second closing delimiter) in order to add an horizontal space (equal to 0.5 em).

```

1696     \@@_patch_preamble_viii_i:n
1697 }

1698 \cs_new_protected:Npn \@@_patch_preamble_viii_i:n #1
1699 {
1700     \bool_lazy_any:nT
1701     {
1702         { \str_if_eq_p:nn { #1 } ( }
1703         { \str_if_eq_p:nn { #1 } [ }
1704         { \str_if_eq_p:nn { #1 } \{ }
1705         { \str_if_eq_p:nn { #1 } \q_stop }
1706     }
1707     { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
1708     \@@_patch_preamble:n #1
1709 }

```

```

1710 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
1711 {
1712   \tl_gput_right:Nn \g_@@_preamble_tl
1713   { ! { \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`.

```

1714   \tl_gput_right:Nx \g_@@_internal_code_after_tl
1715   { \@@_vdottedline:n { \int_use:N \c@jCol } }
1716   \@@_patch_preamble:n
1717 }

```

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.

```

1718 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
1719 {
1720   \str_if_eq:nnTF { #1 } { < }
1721   \@@_patch_preamble_ix:n
1722   {
1723     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1724     {
1725       \tl_gput_right:Nn \g_@@_preamble_tl
1726       { ! { \skip_horizontal:N \arrayrulewidth } }
1727     }
1728     {
1729       \exp_args:NNx
1730       \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
1731       {
1732         \tl_gput_right:Nn \g_@@_preamble_tl
1733         { ! { \skip_horizontal:N \arrayrulewidth } }
1734       }
1735     }
1736     \@@_patch_preamble:n { #1 }
1737   }
1738 }
1739 \cs_new_protected:Npn \@@_patch_preamble_ix:n #1
1740 {
1741   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
1742   \@@_patch_preamble_x:n
1743 }

```

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`).

```

1744 \cs_new_protected:Npn \@@_put_box_in_flow:
1745 {
1746   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
1747   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
1748   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
1749   { \box_use_drop:N \l_tmpa_box }
1750   \@@_put_box_in_flow_i:
1751 }

```

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).

```

1752 \cs_new_protected:Npn \@@_put_box_in_flow_i:
1753 {
1754   \pgfpicture
1755   \@@_qpoint:n { row - 1 }
1756   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1757   \@@_qpoint:n { row - \@@_succ:n \c@iRow }

```

```

1758 \dim_gadd:Nn \g_tmpa_dim \pgf@y
1759 \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).

```

1760 \str_if_in:NnTF \l_@@_baseline_tl { line- }
1761 {
1762   \int_set:Nn \l_tmpa_int
1763   {
1764     \str_range:Nnn
1765       \l_@@_baseline_tl
1766       6
1767     { \tl_count:V \l_@@_baseline_tl }
1768   }
1769   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
1770 }
1771 {
1772   \str_case:VnF \l_@@_baseline_tl
1773   {
1774     { t } { \int_set:Nn \l_tmpa_int 1 }
1775     { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
1776   }
1777   { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
1778   \bool_lazy_or:nnT
1779   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
1780   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
1781   {
1782     \@@_error:n { bad~value~for~baseline }
1783     \int_set:Nn \l_tmpa_int 1
1784   }
1785   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

1786 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
1787 }
1788 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

1789 \endpgfpicture
1790 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
1791 \box_use_drop:N \l_tmpa_box
1792 }

```

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).

```

1793 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
1794 {

```

We need a `{minipage}` because we will insert a LaTeX list for the tabular notes (that means that a `\vtop{\hsize=...}` is not enough).

```

1795 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
1796 \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.

```

1797 \@@_create_extra_nodes:
1798 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
1799 \bool_lazy_or:nnT
1800 { \int_compare_p:nNn \c@tabularnote > 0 }
1801 { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
1802 \@@_insert_tabularnotes:

```

```

1803   \end { minipage }
1804 }
1805 \cs_new_protected:Npn \@@_insert_tabularnotes:
1806 {
1807   \skip_vertical:N 0.65ex

```

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

```

1808   \group_begin:
1809   \l_@@_notes_code_before_tl
1810   \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.

```

1811   \int_compare:nNnT \c@tabularnote > 0
1812   {
1813     \bool_if:NTF \l_@@_notes_para_bool
1814     {
1815       \begin { tabularnotes* }
1816       \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
1817       \end { tabularnotes* }

```

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

```

1818       \par
1819     }
1820   {
1821     \tabularnotes
1822     \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
1823     \endtabularnotes
1824   }
1825 }
1826 \unskip
1827 \group_end:
1828 \bool_if:NT \l_@@_notes_bottomrule_bool
1829 {
1830   \bool_if:NTF \c_@@_booktabs_loaded_bool
1831   {

```

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

```

1832     \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
1833     { \CT@arc@ \hrule height \heavyrulewidth }
1834   }
1835   { \@@_error:n { bottomrule-without-booktabs } }
1836 }
1837 \l_@@_notes_code_after_tl
1838 \seq_gclear:N \g_@@_tabularnotes_seq
1839 \int_gzero:N \c@tabularnote
1840 }

```

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`.

```

1841 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
1842 {
1843   \pgfpicture
1844   \@@_qpoint:n { row - 1 }
1845   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1846   \@@_qpoint:n { row - \int_use:N \c@iRow - base }
1847   \dim_gsub:Nn \g_tmpa_dim \pgf@y
1848   \endpgfpicture
1849   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
1850   \int_compare:nNnT \l_@@_first_row_int = 0

```

```

1851 {
1852   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
1853   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
1854 }
1855 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
1856 }

```

Now, the general case.

```

1857 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
1858 {

```

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

```

1859   \tl_if_eq:NnT \l_@@_baseline_tl { t }
1860   { \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`.

```

1861   \pgfpicture
1862   \@@_qpoint:n { row - 1 }
1863   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
1864   \str_if_in:NnTF \l_@@_baseline_tl { line- }
1865   {
1866     \int_set:Nn \l_tmpa_int
1867     {
1868       \str_range:Nnn
1869       \l_@@_baseline_tl
1870       6
1871       { \tl_count:V \l_@@_baseline_tl }
1872     }
1873     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
1874   }
1875   {
1876     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
1877     \bool_lazy_or:nnT
1878     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
1879     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
1880     {
1881       \@@_error:n { bad~value~for~baseline }
1882       \int_set:Nn \l_tmpa_int 1
1883     }
1884     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
1885   }
1886   \dim_gsub:Nn \g_tmpa_dim \pgf@y
1887   \endpgfpicture
1888   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
1889   \int_compare:nNnT \l_@@_first_row_int = 0
1890   {
1891     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
1892     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
1893   }
1894   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
1895 }

```

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.

```

1896 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
1897 {

```

We will compute the real width of both delimiters used.

```

1898   \dim_zero_new:N \l_@@_real_left_delim_dim
1899   \dim_zero_new:N \l_@@_real_right_delim_dim
1900   \hbox_set:Nn \l_tmpb_box
1901   {

```

```

1902     \c_math_toggle_token
1903     \left #1
1904     \vcenter
1905     {
1906         \vbox_to_ht:nn
1907         { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
1908         { }
1909     }
1910     \right .
1911     \c_math_toggle_token
1912 }
1913 \dim_set:Nn \l_@@_real_left_delim_dim
1914 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
1915 \hbox_set:Nn \l_tmpb_box
1916 {
1917     \c_math_toggle_token
1918     \left .
1919     \vbox_to_ht:nn
1920     { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
1921     { }
1922     \right #2
1923     \c_math_toggle_token
1924 }
1925 \dim_set:Nn \l_@@_real_right_delim_dim
1926 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }

```

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

```

1927     \skip_horizontal:N \l_@@_left_delim_dim
1928     \skip_horizontal:N -\l_@@_real_left_delim_dim
1929     \@@_put_box_in_flow:
1930     \skip_horizontal:N \l_@@_right_delim_dim
1931     \skip_horizontal:N -\l_@@_real_right_delim_dim
1932 }

```

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).

```

1933 \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).

```

1934 {
1935     \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`).

```

1936     { \exp_args:NV \@@_array: \g_@@_preamble_tl }
1937 }
1938 {
1939     \@@_create_col_nodes:
1940     \endarray
1941 }

```

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`).

```

1942 \NewDocumentEnvironment { @@-light-syntax } { b }
1943 {

```

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.

```

1944 \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
1945 \tl_map_inline:nn { #1 }
1946 {
1947   \str_if_eq:nnT { ##1 } { & }
1948   { \@@_fatal:n { ampersand-in-light-syntax } }
1949   \str_if_eq:nnT { ##1 } { \ }
1950   { \@@_fatal:n { double-backslash-in-light-syntax } }
1951 }

```

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`.

```

1952 \@@_light_syntax_i #1 \CodeAfter \q_stop
1953 }

```

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`.

```

1954 { }

1955 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
1956 {
1957   \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).

```

1958 \seq_gclear_new:N \g_@@_rows_seq
1959 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
1960 \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.

```

1961 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1962 { \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`).

```

1963 \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.

```

1964 \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
1965 \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
1966 \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
1967 \@@_create_col_nodes:
1968 \endarray
1969 }

1970 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
1971 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }

1972 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
1973 {
1974   \seq_gclear_new:N \g_@@_cells_seq
1975   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
1976   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
1977   \l_tmpa_tl
1978   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
1979 }

```


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).

```

1980 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
1981 {
1982   \str_if_eq:VnT \g_@@_name_env_str { #2 }
1983   { \@@_fatal:n { empty-environment } }

```

We repeat in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```

1984   \end { #2 }
1985 }

```

The command `\@@_create_col_nodes:` will construct a special last row. That last row is a false row used to create the col nodes and to fix the width of the columns (when the array is constructed with an option which specifies the width of the columns).

```

1986 \cs_new:Npn \@@_create_col_nodes:
1987 {
1988   \crrcr
1989   \int_compare:nNnT \l_@@_first_col_int = 0
1990   {
1991     \omit
1992     \hbox_overlap_left:n
1993     {
1994       \bool_if:NT \l_@@_code_before_bool
1995       { \pgfsys@markposition { \@@_env: - col - 0 } }
1996       \pgfpicture
1997       \pgfrememberpicturepositiononpagetrue
1998       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
1999       \str_if_empty:NF \l_@@_name_str
2000       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2001       \endpgfpicture
2002       \skip_horizontal:N 2\col@sep
2003       \skip_horizontal:N \g_@@_width_first_col_dim
2004     }
2005     &
2006   }
2007   \omit

```

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

```

2008   \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`).

```

2009   \int_compare:nNnTF \l_@@_first_col_int = 0
2010   {
2011     \bool_if:NT \l_@@_code_before_bool
2012     {
2013       \hbox
2014       {
2015         \skip_horizontal:N -0.5\arrayrulewidth
2016         \pgfsys@markposition { \@@_env: - col - 1 }
2017         \skip_horizontal:N 0.5\arrayrulewidth
2018       }
2019     }
2020     \pgfpicture
2021     \pgfrememberpicturepositiononpagetrue
2022     \pgfcoordinate { \@@_env: - col - 1 }
2023     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2024     \str_if_empty:NF \l_@@_name_str
2025     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2026     \endpgfpicture
2027   }
2028   {

```

```

2029 \bool_if:NT \l_@@_code_before_bool
2030 {
2031     \hbox
2032     {
2033         \skip_horizontal:N 0.5\arrayrulewidth
2034         \pgfsys@markposition { \@@_env: - col - 1 }
2035         \skip_horizontal:N -0.5\arrayrulewidth
2036     }
2037 }
2038 \pgfpicture
2039 \pgfrememberpicturepositiononpagetrue
2040 \pgfcoordinate { \@@_env: - col - 1 }
2041 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
2042 \str_if_empty:NF \l_@@_name_str
2043 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2044 \endpgfpicture
2045 }

```

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.

```

2046 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2047 \bool_if:NF \l_@@_auto_columns_width_bool
2048 { \dim_compare:nNt \l_@@_columns_width_dim > \c_zero_dim }
2049 {
2050     \bool_lazy_and:nnTF
2051     \l_@@_auto_columns_width_bool
2052     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2053     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2054     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2055     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2056 }
2057 \skip_horizontal:N \g_tmpa_skip
2058 \hbox
2059 {
2060     \bool_if:NT \l_@@_code_before_bool
2061     {
2062         \hbox
2063         {
2064             \skip_horizontal:N -0.5\arrayrulewidth
2065             \pgfsys@markposition { \@@_env: - col - 2 }
2066             \skip_horizontal:N 0.5\arrayrulewidth
2067         }
2068     }
2069     \pgfpicture
2070     \pgfrememberpicturepositiononpagetrue
2071     \pgfcoordinate { \@@_env: - col - 2 }
2072     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2073     \str_if_empty:NF \l_@@_name_str
2074     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2075     \endpgfpicture
2076 }

```

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.

```

2077 \int_gset:Nn \g_tmpa_int 1
2078 \bool_if:NFT \g_@@_last_col_found_bool
2079 { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
2080 { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
2081 {
2082     &

```

```

2083     \omit
2084     \int_gincr:N \g_tmpa_int

```

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

```

2085     \skip_horizontal:N \g_tmpa_skip
2086     \bool_if:NT \l_@@_code_before_bool
2087     {
2088         \hbox
2089         {
2090             \skip_horizontal:N -0.5\arrayrulewidth
2091             \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2092             \skip_horizontal:N 0.5\arrayrulewidth
2093         }
2094     }

```

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

```

2095     \pgfpicture
2096     \pgfrememberpicturepositiononpagetrue
2097     \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2098     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2099     \str_if_empty:NF \l_@@_name_str
2100     {
2101         \pgfnodealias
2102         { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2103         { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2104     }
2105     \endpgfpicture
2106 }
2107 \bool_if:NT \g_@@_last_col_found_bool
2108 {
2109     \hbox_overlap_right:n
2110     {
2111         % \skip_horizontal:N \col@sep
2112         \skip_horizontal:N \g_@@_width_last_col_dim
2113         \bool_if:NT \l_@@_code_before_bool
2114         {
2115             \pgfsys@markposition
2116             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2117         }
2118         \pgfpicture
2119         \pgfrememberpicturepositiononpagetrue
2120         \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2121         \pgfpointorigin
2122         \str_if_empty:NF \l_@@_name_str
2123         {
2124             \pgfnodealias
2125             { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2126             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2127         }
2128         \endpgfpicture
2129     }
2130 }
2131 \cr
2132 }

```

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

```

2133 \tl_const:Nn \c_@@_preamble_first_col_tl
2134 {
2135     >
2136     {

```

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`).

```

2137 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
2138 \bool_gset_true:N \g_@@_after_col_zero_bool
2139 \@@_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.

```

2140 \hbox_set:Nw \l_@@_cell_box
2141 \@@_math_toggle_token:
2142 \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”.

```

2143 \bool_lazy_and:nnT
2144 { \int_compare_p:nNn \c@iRow > 0 }
2145 {
2146   \bool_lazy_or_p:nn
2147   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2148   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2149 }
2150 {
2151   \l_@@_code_for_first_col_tl
2152   \xglobal \colorlet { nicematrix-first-col } { . }
2153 }
2154 }

```

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`.

```

2155 l
2156 <
2157 {
2158   \@@_math_toggle_token:
2159   \hbox_set_end:
2160   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2161   \@@_adjust_size_box:
2162   \@@_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.

```

2163 \dim_gset:Nn \g_@@_width_first_col_dim
2164 { \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.

```

2165 \hbox_overlap_left:n
2166 {
2167   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2168   \@@_node_for_the_cell:
2169   { \box_use_drop:N \l_@@_cell_box }
2170   \skip_horizontal:N \l_@@_left_delim_dim
2171   \skip_horizontal:N \l_@@_left_margin_dim
2172   \skip_horizontal:N \l_@@_extra_left_margin_dim
2173 }
2174 \bool_gset_false:N \g_@@_empty_cell_bool
2175 \skip_horizontal:N -2\col@sep
2176 }
2177 }

```

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

```

2178 \tl_const:Nn \c_@@_preamble_last_col_tl
2179 {
2180   >
2181   {

```

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`).

```

2182 \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.

```

2183 \bool_gset_true:N \g_@@_last_col_found_bool
2184 \int_gincr:N \c@jCol
2185 \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.

```

2186 \hbox_set:Nw \l_@@_cell_box
2187 \@@_math_toggle_token:
2188 \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”.

```

2189 \int_compare:nNnT \c@iRow > 0
2190 {
2191   \bool_lazy_or:nnT
2192     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2193     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2194     {
2195       \l_@@_code_for_last_col_tl
2196       \xglobal \colorlet { nicematrix-last-col } { . }
2197     }
2198   }
2199 }
2200 1
2201 <
2202 {
2203   \@@_math_toggle_token:
2204   \hbox_set_end:
2205   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2206   \@@_adjust_size_box:
2207   \@@_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.

```

2208 \dim_gset:Nn \g_@@_width_last_col_dim
2209 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2210 \skip_horizontal:N -2\col@sep

```

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

```

2211 \hbox_overlap_right:n
2212 {
2213   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2214   {
2215     \skip_horizontal:N \l_@@_right_delim_dim
2216     \skip_horizontal:N \l_@@_right_margin_dim
2217     \skip_horizontal:N \l_@@_extra_right_margin_dim
2218     \@@_node_for_the_cell:
2219   }
2220 }
2221 \bool_gset_false:N \g_@@_empty_cell_bool
2222 }
2223 }

```

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.

```

2224 \NewDocumentEnvironment { NiceArray } { }
2225 {
2226   \bool_set_true:N \l_@@_NiceArray_bool
2227   \str_if_empty:NT \g_@@_name_env_str
2228     { \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).

```

2229 \NiceArrayWithDelims . .
2230 }
2231 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

2232 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2233 {
2234   \NewDocumentEnvironment { #1 NiceArray } { }
2235   {
2236     \str_if_empty:NT \g_@@_name_env_str
2237     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2238     \@@_test_if_math_mode:
2239     \NiceArrayWithDelims #2 #3
2240   }
2241   { \endNiceArrayWithDelims }
2242 }
2243 \@@_def_env:nnn p ( )
2244 \@@_def_env:nnn b [ ]
2245 \@@_def_env:nnn B \{ \}
2246 \@@_def_env:nnn v | |
2247 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2248 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2249 {
2250   \bool_set_true:N \l_@@_Matrix_bool
2251   \use:c { #1 NiceArray }
2252   {
2253     *
2254     {
2255       \int_compare:nNnTF \l_@@_last_col_int < 0
2256       \c@MaxMatrixCols
2257       { \@@_pred:n \l_@@_last_col_int }
2258     }
2259     { > \@@_Cell: #2 < \@@_end_Cell: }
2260   }
2261 }
2262 \clist_map_inline:nn { { } , p , b , B , v , V }
2263 {
2264   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2265   {
2266     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2267     \tl_set:Nn \l_@@_type_of_col_tl c
2268     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2269     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2270   }
2271   { \use:c { end #1 NiceArray } }
2272 }

```

The following command will be linked to \NotEmpty in the environments of nicematrix.

```

2273 \cs_new_protected:Npn \@@_NotEmpty:
2274 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

The environments {NiceTabular} and {NiceTabular*}

```

2275 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2276 {
2277   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2278   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2279   \bool_set_true:N \l_@@_NiceTabular_bool
2280   \NiceArray { #2 }
2281 }
2282 { \endNiceArray }

2283 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
2284 {
2285   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2286   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
2287   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2288   \bool_set_true:N \l_@@_NiceTabular_bool
2289   \NiceArray { #3 }
2290 }
2291 { \endNiceArray }

```

After the construction of the array

```

2292 \cs_new_protected:Npn \@@_after_array:
2293 {
2294   \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.

```

2295   \bool_if:NT \g_@@_last_col_found_bool
2296   { \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`.

```

2297   \bool_if:NT \l_@@_last_col_without_value_bool
2298   {
2299     \dim_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int
2300     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2301     \iow_shipout:Nx \@mainaux
2302     {
2303       \cs_gset:cpn { @@_last_col_ \int_use:N \g_@@_env_int }
2304       { \int_use:N \g_@@_col_total_int }
2305     }
2306     \str_if_empty:NF \l_@@_name_str
2307     {
2308       \iow_shipout:Nx \@mainaux
2309       {
2310         \cs_gset:cpn { @@_last_col_ \l_@@_name_str }
2311         { \int_use:N \g_@@_col_total_int }
2312       }
2313     }
2314     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2315   }

```

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.

```

2316   \bool_if:NT \l_@@_last_row_without_value_bool
2317   {
2318     \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.

```

2319 \bool_if:NF \l_@@_light_syntax_bool
2320 {
2321   \iow_shipout:Nn \@mainaux \ExplSyntaxOn
2322   \iow_shipout:Nx \@mainaux
2323   {
2324     \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int }
2325     { \int_use:N \g_@@_row_total_int }
2326   }

```

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.

```

2327   \str_if_empty:NF \l_@@_name_str
2328   {
2329     \iow_shipout:Nx \@mainaux
2330     {
2331       \cs_gset:cpn { @@_last_row_ \l_@@_name_str }
2332       { \int_use:N \g_@@_row_total_int }
2333     }
2334   }
2335   \iow_shipout:Nn \@mainaux \ExplSyntaxOff
2336 }
2337 }

```

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

```

2338 \bool_if:NT \l_@@_code_before_bool
2339 {
2340   \iow_now:Nn \@mainaux \ExplSyntaxOn
2341   \iow_now:Nx \@mainaux
2342   { \seq_clear_new:c { @@_size _ \int_use:N \g_@@_env_int _ seq } }
2343   \iow_now:Nx \@mainaux
2344   {
2345     \seq_gset_from_clist:cn { @@_size _ \int_use:N \g_@@_env_int _ seq }
2346     {
2347       \int_use:N \l_@@_first_row_int ,
2348       \int_use:N \g_@@_row_total_int ,
2349       \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`.

```

2350     \bool_lazy_and:nnTF
2351     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
2352     { \bool_not_p:n \g_@@_last_col_found_bool }
2353     \@@_succ:n
2354     \int_use:N
2355     \g_@@_col_total_int
2356   }

```

We write also the potential content of `\g_@@_pos_of_blocks_seq` (it will be useful if the commands `\rowcolors` is used with the key `respect-blocks`).

```

2357     \seq_gset_from_clist:cn
2358     { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
2359     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
2360   }
2361   \iow_now:Nn \@mainaux \ExplSyntaxOff
2362 }

2363 \@@_create_diag_nodes:
2364 \str_if_empty:NF \l_@@_name_str
2365 {
2366   \pgfpicture
2367   \pgfrememberpicturepositiononpagetrue
2368   \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 }
2369   \endpgfpicture
2370 }

```


By default, the diagonal lines will be parallelized⁵¹. 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.

```

2371 \bool_if:NT \l_@@_parallelize_diags_bool
2372 {
2373     \int_gzero_new:N \g_@@_ddots_int
2374     \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.

```

2375     \dim_gzero_new:N \g_@@_delta_x_one_dim
2376     \dim_gzero_new:N \g_@@_delta_y_one_dim
2377     \dim_gzero_new:N \g_@@_delta_x_two_dim
2378     \dim_gzero_new:N \g_@@_delta_y_two_dim
2379 }
2380 \int_zero_new:N \l_@@_initial_i_int
2381 \int_zero_new:N \l_@@_initial_j_int
2382 \int_zero_new:N \l_@@_final_i_int
2383 \int_zero_new:N \l_@@_final_j_int
2384 \bool_set_false:N \l_@@_initial_open_bool
2385 \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_radius_dim` and `\l_@@_inter_dots_dim` (used to draw the dotted lines created by `\hdottedline` and `\vdotteline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.

```

2386 \bool_if:NT \l_@@_small_bool
2387 {
2388     \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
2389     \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.

```

2390     \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
2391 }

```

Now, we actually draw the dotted lines (specified by `\Cdots`, `\Vdots`, etc.).

```

2392 \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it will do nothing.

```

2393 \@@_compute_corners:

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-}`).

```

2394 \@@_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`.

```

2395 \bool_lazy_all:nT
2396 {
2397     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
2398     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
2399     { \seq_if_empty_p:N \l_@@_empty_corner_cells_seq }
2400 }
2401 {
2402     \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn

```

⁵¹It's possible to use the option `parallelize-diags` to disable this parallelization.

```

2403     \cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn
2404   }
2405   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
2406   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
2407   \cs_set_eq:NN \SubMatrix \@@_SubMatrix

```

Now, the internal code-after and then, the \CodeAfter.

```

2408   \bool_if:NT \c_@@_tikz_loaded_bool
2409   {
2410     \tikzset
2411     {
2412       every~picture / .style =
2413       {
2414         overlay ,
2415         remember~picture ,
2416         name~prefix = \@@_env: -
2417       }
2418     }
2419   }
2420   \cs_set_eq:NN \line \@@_line
2421   \g_@@_internal_code_after_tl
2422   \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.

```

2423   \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).

```

2424   \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:.

```

2425   \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
2426   \scan_stop:
2427   \tl_gclear:N \g_nicematrix_code_after_tl
2428   \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.

```

2429   \tl_if_empty:NF \g_nicematrix_code_before_tl
2430   {

```

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.

```

2431     \cs_set_protected:Npn \rectanglecolor { }
2432     \cs_set_protected:Npn \columncolor { }
2433     \iow_now:Nn \@mainaux \ExplSyntaxOn
2434     \iow_now:Nx \@mainaux
2435     {
2436       \tl_gset:cn
2437       { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
2438       { \exp_not:V \g_nicematrix_code_before_tl }
2439     }
2440     \iow_now:Nn \@mainaux \ExplSyntaxOff
2441     \bool_set_true:N \l_@@_code_before_bool
2442   }

```

```

2443   \str_gclear:N \g_@@_name_env_str
2444   \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁵². 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`.

```
2445 \cs_gset_eq:NN \CT@arc@ \@_old_CT@arc@
2446 }
```

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`).

```
2447 \NewDocumentCommand \@_CodeAfter_keys: { 0 { } }
2448 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }
```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It's possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```
2449 \cs_new_protected:Npn \@_adjust_pos_of_blocks_seq:
2450 {
2451   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
2452   { \@_adjust_pos_of_blocks_seq_i:nnnn ##1 }
2453 }
```

The following command must *not* be protected.

```
2454 \cs_new:Npn \@_adjust_pos_of_blocks_seq_i:nnnn #1 #2 #3 #4
2455 {
2456   { #1 }
2457   { #2 }
2458   {
2459     \int_compare:nNnTF { #3 } > { 99 }
2460     { \int_use:N \c@iRow }
2461     { #3 }
2462   }
2463   {
2464     \int_compare:nNnTF { #4 } > { 99 }
2465     { \int_use:N \c@jCol }
2466     { #4 }
2467   }
2468 }
```

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).

```
2469 \AtBeginDocument
2470 {
2471   \cs_new_protected:Npx \@_draw_dotted_lines:
2472   {
2473     \c_@@_pgfortikzpicture_tl
2474     \@_draw_dotted_lines_i:
2475     \c_@@_endpgfortikzpicture_tl
2476   }
2477 }
```

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

```
2478 \cs_new_protected:Npn \@_draw_dotted_lines_i:
2479 {
2480   \pgfrememberpicturepositiononpagetrue
```

⁵²e.g. `\color[rgb]{0.5,0.5,0}`

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

```
2509 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
2510 \int_set:Nn \l_@@_initial_i_int { #1 }
2511 \int_set:Nn \l_@@_initial_j_int { #2 }
2512 \int_set:Nn \l_@@_final_i_int { #1 }
2513 \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.

```
2514 \bool_set_false:N \l_@@_stop_loop_bool
2515 \bool_do_until:Nn \l_@@_stop_loop_bool
2516 {
2517   \int_add:Nn \l_@@_final_i_int { #3 }
2518   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
2519   \bool_set_false:N \l_@@_final_open_bool
2520   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
2521   {
2522     \int_compare:nNnTF { #3 } = 1
2523     { \bool_set_true:N \l_@@_final_open_bool }
2524     {
2525       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
2526       { \bool_set_true:N \l_@@_final_open_bool }
2527     }
2528   }
2529   {
2530     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
2531     {
2532       \int_compare:nNnT { #4 } = { -1 }
2533       { \bool_set_true:N \l_@@_final_open_bool }
2534     }
2535     {
2536       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
2537       {
2538         \int_compare:nNnT { #4 } = 1
2539         { \bool_set_true:N \l_@@_final_open_bool }
2540       }
2541     }
2542   }
2543   \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.

```
2544   {
```

We do a step backwards.

```
2545     \int_sub:Nn \l_@@_final_i_int { #3 }
2546     \int_sub:Nn \l_@@_final_j_int { #4 }
2547     \bool_set_true:N \l_@@_stop_loop_bool
2548   }
```

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`.

```
2549   {
2550     \cs_if_exist:cTF
2551     {
2552       @@ _ dotted _
2553       \int_use:N \l_@@_final_i_int -
2554       \int_use:N \l_@@_final_j_int
2555     }
2556     {
2557       \int_sub:Nn \l_@@_final_i_int { #3 }
```

```

2558     \int_sub:Nn \l_@@_final_j_int { #4 }
2559     \bool_set_true:N \l_@@_final_open_bool
2560     \bool_set_true:N \l_@@_stop_loop_bool
2561   }
2562   {
2563     \cs_if_exist:cTF
2564     {
2565       pgf @ sh @ ns @ \@@_env:
2566       - \int_use:N \l_@@_final_i_int
2567       - \int_use:N \l_@@_final_j_int
2568     }
2569     { \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.

```

2570     {
2571       \cs_set:cpn
2572       {
2573         @@ _ dotted _
2574         \int_use:N \l_@@_final_i_int -
2575         \int_use:N \l_@@_final_j_int
2576       }
2577     { }
2578   }
2579 }
2580 }
2581 }

```

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

```

2582 \bool_set_false:N \l_@@_stop_loop_bool
2583 \bool_do_until:Nn \l_@@_stop_loop_bool
2584 {
2585   \int_sub:Nn \l_@@_initial_i_int { #3 }
2586   \int_sub:Nn \l_@@_initial_j_int { #4 }
2587   \bool_set_false:N \l_@@_initial_open_bool
2588   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
2589   {
2590     \int_compare:nNnTF { #3 } = 1
2591     { \bool_set_true:N \l_@@_initial_open_bool }
2592     {
2593       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
2594       { \bool_set_true:N \l_@@_initial_open_bool }
2595     }
2596   }
2597   {
2598     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
2599     {
2600       \int_compare:nNnT { #4 } = 1
2601       { \bool_set_true:N \l_@@_initial_open_bool }
2602     }
2603     {
2604       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
2605       {
2606         \int_compare:nNnT { #4 } = { -1 }
2607         { \bool_set_true:N \l_@@_initial_open_bool }
2608       }
2609     }
2610   }

```

```

2611 \bool_if:NTF \l_@@_initial_open_bool
2612 {
2613   \int_add:Nn \l_@@_initial_i_int { #3 }
2614   \int_add:Nn \l_@@_initial_j_int { #4 }
2615   \bool_set_true:N \l_@@_stop_loop_bool
2616 }
2617 {
2618   \cs_if_exist:cTF
2619   {
2620     @@ _ dotted _
2621     \int_use:N \l_@@_initial_i_int -
2622     \int_use:N \l_@@_initial_j_int
2623   }
2624   {
2625     \int_add:Nn \l_@@_initial_i_int { #3 }
2626     \int_add:Nn \l_@@_initial_j_int { #4 }
2627     \bool_set_true:N \l_@@_initial_open_bool
2628     \bool_set_true:N \l_@@_stop_loop_bool
2629   }
2630   {
2631     \cs_if_exist:cTF
2632     {
2633       pgf @ sh @ ns @ \@@_env:
2634       - \int_use:N \l_@@_initial_i_int
2635       - \int_use:N \l_@@_initial_j_int
2636     }
2637     { \bool_set_true:N \l_@@_stop_loop_bool }
2638     {
2639       \cs_set:cpn
2640       {
2641         @@ _ dotted _
2642         \int_use:N \l_@@_initial_i_int -
2643         \int_use:N \l_@@_initial_j_int
2644       }
2645       { }
2646     }
2647   }
2648 }
2649 }

```

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.

```

2650 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
2651 {
2652   { \int_use:N \l_@@_initial_i_int }
2653   { \int_use:N \l_@@_initial_j_int }
2654   { \int_use:N \l_@@_final_i_int }
2655   { \int_use:N \l_@@_final_j_int }
2656 }
2657 }

```

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).

```

2658 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
2659 {
2660   \int_set:Nn \l_@@_row_min_int 1
2661   \int_set:Nn \l_@@_col_min_int 1
2662   \int_set_eq:NN \l_@@_row_max_int \c@iRow
2663   \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`.

```

2664 \seq_map_inline:Nn \g_@@_submatrix_seq
2665 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
2666 }

```

#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.

```

2667 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
2668 {
2669   \bool_if:nT
2670   {
2671     \int_compare_p:n { #3 <= #1 }
2672     && \int_compare_p:n { #1 <= #5 }
2673     && \int_compare_p:n { #4 <= #2 }
2674     && \int_compare_p:n { #2 <= #6 }
2675   }
2676   {
2677     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
2678     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
2679     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
2680     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
2681   }
2682 }

2683 \cs_new_protected:Npn \@@_set_initial_coords:
2684 {
2685   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2686   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2687 }
2688 \cs_new_protected:Npn \@@_set_final_coords:
2689 {
2690   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2691   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2692 }
2693 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
2694 {
2695   \pgfpointanchor
2696   {
2697     \@@_env:
2698     - \int_use:N \l_@@_initial_i_int
2699     - \int_use:N \l_@@_initial_j_int
2700   }
2701   { #1 }
2702   \@@_set_initial_coords:
2703 }
2704 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
2705 {
2706   \pgfpointanchor
2707   {
2708     \@@_env:
2709     - \int_use:N \l_@@_final_i_int
2710     - \int_use:N \l_@@_final_j_int
2711   }
2712   { #1 }
2713   \@@_set_final_coords:
2714 }

2715 \cs_new_protected:Npn \@@_open_x_initial_dim:
2716 {
2717   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
2718   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2719   {
2720     \cs_if_exist:cT
2721     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }

```



```

2722     {
2723         \pgfpointanchor
2724         { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
2725         { west }
2726         \dim_set:Nn \l_@@_x_initial_dim
2727         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
2728     }
2729 }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

2730     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
2731     {
2732         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2733         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2734         \dim_add:Nn \l_@@_x_initial_dim \col@sep
2735     }
2736 }
2737 \cs_new_protected:Npn \@@_open_x_final_dim:
2738 {
2739     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
2740     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2741     {
2742         \cs_if_exist:cT
2743         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
2744         {
2745             \pgfpointanchor
2746             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
2747             { east }
2748             \dim_set:Nn \l_@@_x_final_dim
2749             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
2750         }
2751     }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

2752     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
2753     {
2754         \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2755         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2756         \dim_sub:Nn \l_@@_x_final_dim \col@sep
2757     }
2758 }

```

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.

```

2759 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
2760 {
2761     \@@_adjust_to_submatrix:nn { #1 } { #2 }
2762     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2763     {
2764         \@@_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.

```

2765     \group_begin:
2766     \int_compare:nNnTF { #1 } = 0
2767     { \color { nicematrix-first-row } }
2768     {

```

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.

```

2769         \int_compare:nNnT { #1 } = \l_@@_last_row_int
2770         { \color { nicematrix-last-row } }
2771     }

```

```

2772         \keys_set:nn { NiceMatrix / xdots } { #3 }
2773         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2774         \@@_actually_draw_Ldots:
2775     \group_end:
2776 }
2777 }

```

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`.

```

2778 \cs_new_protected:Npn \@@_actually_draw_Ldots:
2779 {
2780     \bool_if:NTF \l_@@_initial_open_bool
2781     {
2782         \@@_open_x_initial_dim:
2783         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
2784         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
2785     }
2786     { \@@_set_initial_coords_from_anchor:n { base-east } }
2787     \bool_if:NTF \l_@@_final_open_bool
2788     {
2789         \@@_open_x_final_dim:
2790         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
2791         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
2792     }
2793     { \@@_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 (?).

```

2794     \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
2795     \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
2796     \@@_draw_line:
2797 }

```

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.

```

2798 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
2799 {
2800     \@@_adjust_to_submatrix:nn { #1 } { #2 }
2801     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2802     {
2803         \@@_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.

```

2804     \group_begin:
2805     \int_compare:nNnTF { #1 } = 0
2806     { \color { nicematrix-first-row } }
2807     {

```

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.

```

2808         \int_compare:nNnT { #1 } = \l_@@_last_row_int
2809         { \color { nicematrix-last-row } }
2810     }
2811     \keys_set:nn { NiceMatrix / xdots } { #3 }
2812     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2813     \@@_actually_draw_Cdots:
2814 \group_end:
2815 }
2816 }

```

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.`

```

2817 \cs_new_protected:Npn \@@_actually_draw_Cdots:
2818 {
2819     \bool_if:NTF \l_@@_initial_open_bool
2820     { \@@_open_x_initial_dim: }
2821     { \@@_set_initial_coords_from_anchor:n { mid-east } }
2822     \bool_if:NTF \l_@@_final_open_bool
2823     { \@@_open_x_final_dim: }
2824     { \@@_set_final_coords_from_anchor:n { mid-west } }
2825     \bool_lazy_and:nnTF
2826     \l_@@_initial_open_bool
2827     \l_@@_final_open_bool
2828     {
2829         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
2830         \dim_set_eq:NN \l_tmpa_dim \pgf@y
2831         \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
2832         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
2833         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
2834     }
2835     {
2836         \bool_if:NT \l_@@_initial_open_bool
2837         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
2838         \bool_if:NT \l_@@_final_open_bool
2839         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
2840     }
2841     \@@_draw_line:
2842 }
2843 \cs_new_protected:Npn \@@_open_y_initial_dim:
2844 {
2845     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
2846     \dim_set:Nn \l_@@_y_initial_dim
2847     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
2848     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2849     {
2850         \cs_if_exist:cT
2851         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
2852         {
2853             \pgfpointanchor
2854             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }

```

```

2855         { north }
2856         \dim_set:Nn \l_@@_y_initial_dim
2857         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
2858     }
2859 }
2860 }
2861 \cs_new_protected:Npn \@@_open_y_final_dim:
2862 {
2863     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
2864     \dim_set:Nn \l_@@_y_final_dim
2865     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
2866     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2867     {
2868         \cs_if_exist:cT
2869         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
2870         {
2871             \pgfpointanchor
2872             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
2873             { south }
2874             \dim_set:Nn \l_@@_y_final_dim
2875             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
2876         }
2877     }
2878 }

```

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.

```

2879 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
2880 {
2881     \@@_adjust_to_submatrix:nn { #1 } { #2 }
2882     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2883     {
2884         \@@_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.

```

2885     \group_begin:
2886     \int_compare:nNnTF { #2 } = 0
2887     { \color { nicematrix-first-col } }
2888     {
2889         \int_compare:nNnT { #2 } = \l_@@_last_col_int
2890         { \color { nicematrix-last-col } }
2891     }
2892     \keys_set:nn { NiceMatrix / xdots } { #3 }
2893     \tl_if_empty:VF \l_@@_xdots_color_tl
2894     { \color { \l_@@_xdots_color_tl } }
2895     \@@_actually_draw_Vdots:
2896     \group_end:
2897 }
2898 }

```

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.

```
2899 \cs_new_protected:Npn \@@_actually_draw_Vdots:
2900 {
```

The boolean \l_tmpa_bool indicates whether the column is of type l or may be considered as if.

```
2901 \bool_set_false:N \l_tmpa_bool
```

First the case when the line is closed on both ends.

```
2902 \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
2903 {
2904   \@@_set_initial_coords_from_anchor:n { south-west }
2905   \@@_set_final_coords_from_anchor:n { north-west }
2906   \bool_set:Nn \l_tmpa_bool
2907   { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
2908 }
```

Now, we try to determine whether the column is of type c or may be considered as if.

```
2909 \bool_if:NTF \l_@@_initial_open_bool
2910 \@@_open_y_initial_dim:
2911 { \@@_set_initial_coords_from_anchor:n { south } }
2912 \bool_if:NTF \l_@@_final_open_bool
2913 \@@_open_y_final_dim:
2914 { \@@_set_final_coords_from_anchor:n { north } }
2915 \bool_if:NTF \l_@@_initial_open_bool
2916 {
2917   \bool_if:NTF \l_@@_final_open_bool
2918   {
2919     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2920     \dim_set_eq:NN \l_tmpa_dim \pgf@x
2921     \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
2922     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
2923     \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.

```
2924 \int_compare:nNnT \l_@@_last_col_int > { -2 }
2925 {
2926   \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
2927   {
2928     \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
2929     \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
2930     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
2931     \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
2932   }
2933 }
2934 }
2935 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
2936 }
2937 {
2938   \bool_if:NTF \l_@@_final_open_bool
2939   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
2940 }
```

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.

```
2941 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
2942 {
2943   \dim_set:Nn \l_@@_x_initial_dim
2944   {
2945     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
2946     \l_@@_x_initial_dim \l_@@_x_final_dim
2947   }
2948   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
2949 }
```

```

2950     }
2951   }
2952   \@@_draw_line:
2953 }

```

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.

```

2954 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
2955 {
2956   \@@_adjust_to_submatrix:nn { #1 } { #2 }
2957   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
2958   {
2959     \@@_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.

```

2960     \group_begin:
2961     \keys_set:nn { NiceMatrix / xdots } { #3 }
2962     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
2963     \@@_actually_draw_Ddots:
2964   \group_end:
2965 }
2966 }

```

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`.

```

2967 \cs_new_protected:Npn \@@_actually_draw_Ddots:
2968 {
2969   \bool_if:NTF \l_@@_initial_open_bool
2970   {
2971     \@@_open_y_initial_dim:
2972     % \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
2973     % \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
2974     \@@_open_x_initial_dim:
2975   }
2976   { \@@_set_initial_coords_from_anchor:n { south-east } }
2977   \bool_if:NTF \l_@@_final_open_bool
2978   {
2979     % \@@_open_y_final_dim:
2980     % \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
2981     \@@_open_x_final_dim:
2982     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
2983   }
2984   { \@@_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.

```

2985   \bool_if:NT \l_@@_parallelize_diags_bool
2986   {
2987     \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).

```
2988 \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.

```
2989 {
2990   \dim_gset:Nn \g_@@_delta_x_one_dim
2991   { \l_@@_x_final_dim - \l_@@_x_initial_dim }
2992   \dim_gset:Nn \g_@@_delta_y_one_dim
2993   { \l_@@_y_final_dim - \l_@@_y_initial_dim }
2994 }
```

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`.

```
2995 {
2996   \dim_set:Nn \l_@@_y_final_dim
2997   {
2998     \l_@@_y_initial_dim +
2999     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3000     \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3001   }
3002 }
3003 }
3004 \@@_draw_line:
3005 }
```

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.

```
3006 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3007 {
3008   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3009   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3010   {
3011     \@@_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.

```
3012   \group_begin:
3013   \keys_set:nn { NiceMatrix / xdots } { #3 }
3014   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3015   \@@_actually_draw_Iddots:
3016   \group_end:
3017 }
3018 }
```

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`.

```

3019 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3020 {
3021   \bool_if:NTF \l_@@_initial_open_bool
3022   {
3023     % \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3024     % \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3025     \@@_open_y_initial_dim:
3026     % \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3027     % \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3028     \@@_open_x_initial_dim:
3029   }
3030   { \@@_set_initial_coords_from_anchor:n { south-west } }
3031   \bool_if:NTF \l_@@_final_open_bool
3032   {
3033     % \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
3034     % \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3035     \@@_open_y_final_dim:
3036     % \@@_qpoint:n { col - \int_use:N \l_@@_final_j_int }
3037     % \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3038     \@@_open_x_final_dim:
3039   }
3040   { \@@_set_final_coords_from_anchor:n { north-east } }
3041   \bool_if:NT \l_@@_parallelize_diags_bool
3042   {
3043     \int_gincr:N \g_@@_iddots_int
3044     \int_compare:nNnTF \g_@@_iddots_int = 1
3045     {
3046       \dim_gset:Nn \g_@@_delta_x_two_dim
3047       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3048       \dim_gset:Nn \g_@@_delta_y_two_dim
3049       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3050     }
3051     {
3052       \dim_set:Nn \l_@@_y_final_dim
3053       {
3054         \l_@@_y_initial_dim +
3055         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3056         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3057       }
3058     }
3059   }
3060   \@@_draw_line:
3061 }

```

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`

```

3062 \cs_new_protected:Npn \@@_draw_line:
3063 {

```



```

3064 \pgfrememberpicturepositiononpagetrue
3065 \pgf@relevantforpicturesizefalse
3066 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
3067   \@@_draw_standard_dotted_line:
3068   \@@_draw_non_standard_dotted_line:
3069 }

```

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.

```

3070 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
3071 {
3072   \begin { scope }
3073   \exp_args:No \@@_draw_non_standard_dotted_line:n
3074     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3075 }

```

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.

```

3076 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
3077 {
3078   \@@_draw_non_standard_dotted_line:nVV
3079   { #1 }
3080   \l_@@_xdots_up_tl
3081   \l_@@_xdots_down_tl
3082 }

3083 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:nnn #1 #2 #3
3084 {
3085   \draw
3086   [
3087     #1 ,
3088     shorten-> = \l_@@_xdots_shorten_dim ,
3089     shorten-< = \l_@@_xdots_shorten_dim ,
3090   ]
3091     ( \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).

```

3092   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3093   node [ sloped , below ] { $ \scriptstyle #3 $ }
3094   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3095   \end { scope }
3096 }
3097 \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).

```

3098 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3099 {
3100   \bool_lazy_and:nnF
3101     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3102     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3103     {
3104       \pgfscope
3105       \pgftransformshift
3106       {
3107         \pgfpointlineattime { 0.5 }
3108         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3109         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3110       }
3111       \pgftransformrotate
3112       {

```

```

3113         \fp_eval:n
3114         {
3115             atand
3116             (
3117                 \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3118                 \l_@@_x_final_dim - \l_@@_x_initial_dim
3119             )
3120         }
3121     }
3122     \pgfnode
3123     { rectangle }
3124     { south }
3125     {
3126         \c_math_toggle_token
3127         \scriptstyle \l_@@_xdots_up_tl
3128         \c_math_toggle_token
3129     }
3130     { }
3131     { \pgfusepath { } }
3132     \pgfnode
3133     { rectangle }
3134     { north }
3135     {
3136         \c_math_toggle_token
3137         \scriptstyle \l_@@_xdots_down_tl
3138         \c_math_toggle_token
3139     }
3140     { }
3141     { \pgfusepath { } }
3142     \endpgfscope
3143 }
3144 \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.

```

3145     \dim_zero_new:N \l_@@_l_dim
3146     \dim_set:Nn \l_@@_l_dim
3147     {
3148         \fp_to_dim:n
3149         {
3150             sqrt
3151             (
3152                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3153                 +
3154                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3155             )
3156         }
3157     }

```

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.

```

3158     \bool_lazy_or:nnF
3159     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3160     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3161     \@@_draw_standard_dotted_line_i:
3162 \group_end:
3163 }
3164 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3165 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3166 {

```

The number of dots will be $\backslash l_tmpa_int + 1$.

```

3167   \bool_if:NTF \l_@@_initial_open_bool
3168   {
3169     \bool_if:NTF \l_@@_final_open_bool
3170     {
3171       \int_set:Nn \l_tmpa_int
3172       { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3173     }
3174     {
3175       \int_set:Nn \l_tmpa_int
3176       {
3177         \dim_ratio:nn
3178         { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3179         \l_@@_inter_dots_dim
3180       }
3181     }
3182   }
3183   {
3184     \bool_if:NTF \l_@@_final_open_bool
3185     {
3186       \int_set:Nn \l_tmpa_int
3187       {
3188         \dim_ratio:nn
3189         { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3190         \l_@@_inter_dots_dim
3191       }
3192     }
3193     {
3194       \int_set:Nn \l_tmpa_int
3195       {
3196         \dim_ratio:nn
3197         { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3198         \l_@@_inter_dots_dim
3199       }
3200     }
3201   }

```

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

```

3202   \dim_set:Nn \l_tmpa_dim
3203   {
3204     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3205     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3206   }
3207   \dim_set:Nn \l_tmpb_dim
3208   {
3209     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3210     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3211   }

```

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 $\backslash l_tmpb_int$.

```

3212   \int_set:Nn \l_tmpb_int
3213   {
3214     \bool_if:NTF \l_@@_initial_open_bool
3215     { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3216     { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3217   }

```

In the loop over the dots, the dimensions $\backslash l_@@_x_initial_dim$ and $\backslash l_@@_y_initial_dim$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

3218 \dim_gadd:Nn \l_@@_x_initial_dim
3219 {
3220   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3221   \dim_ratio:nn
3222   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3223   { 2 \l_@@_l_dim }
3224   * \l_tmpb_int
3225 }
3226 \dim_gadd:Nn \l_@@_y_initial_dim
3227 {
3228   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3229   \dim_ratio:nn
3230   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3231   { 2 \l_@@_l_dim }
3232   * \l_tmpb_int
3233 }
3234 \pgf@relevantforpicturesizefalse
3235 \int_step_inline:nnn 0 \l_tmpa_int
3236 {
3237   \pgfpathcircle
3238   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3239   { \l_@@_radius_dim }
3240   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3241   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3242 }
3243 \pgfusepathqfill
3244 }

```

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.

```

3245 \AtBeginDocument
3246 {
3247   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3248   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3249   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3250   {
3251     \int_compare:nNnTF \c@jCol = 0
3252     { \@@_error:nn { in~first~col } \Ldots }
3253     {
3254       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3255       { \@@_error:nn { in~last~col } \Ldots }
3256       {
3257         \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
3258         { #1 , down = #2 , up = #3 }
3259       }
3260     }
3261     \bool_if:NF \l_@@_nullify_dots_bool
3262     { \phantom { \ensuremath { \@@_old_ldots } } }

```

```

3263     \bool_gset_true:N \g_@@_empty_cell_bool
3264 }

\exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3265 {
3266     \int_compare:nNnTF \c@jCol = 0
3267     { \@@_error:nn { in~first~col } \Cdots }
3268     {
3269         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3270         { \@@_error:nn { in~last~col } \Cdots }
3271         {
3272             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
3273             { #1 , down = #2 , up = #3 }
3274         }
3275     }
3276 }
3277 \bool_if:NF \l_@@_nullify_dots_bool
3278 { \phantom { \ensuremath { \@@_old_cdots } } }
3279 \bool_gset_true:N \g_@@_empty_cell_bool
3280 }

\exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
3281 {
3282     \int_compare:nNnTF \c@iRow = 0
3283     { \@@_error:nn { in~first~row } \Vdots }
3284     {
3285         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
3286         { \@@_error:nn { in~last~row } \Vdots }
3287         {
3288             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
3289             { #1 , down = #2 , up = #3 }
3290         }
3291     }
3292 }
3293 \bool_if:NF \l_@@_nullify_dots_bool
3294 { \phantom { \ensuremath { \@@_old_vdots } } }
3295 \bool_gset_true:N \g_@@_empty_cell_bool
3296 }

\exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
3297 {
3298     \int_case:nnF \c@iRow
3299     {
3300         0 { \@@_error:nn { in~first~row } \Ddots }
3301         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
3302     }
3303     {
3304         \int_case:nnF \c@jCol
3305         {
3306             0 { \@@_error:nn { in~first~col } \Ddots }
3307             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
3308         }
3309         {
3310             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3311             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
3312             { #1 , down = #2 , up = #3 }
3313         }
3314     }
3315 }
3316 }
3317 \bool_if:NF \l_@@_nullify_dots_bool
3318 { \phantom { \ensuremath { \@@_old_ddots } } }
3319 \bool_gset_true:N \g_@@_empty_cell_bool
3320 }

```

```

3321 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
3322 {
3323   \int_case:nnF \c@iRow
3324   {
3325     0 { \@@_error:nn { in~first~row } \Iddots }
3326     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
3327   }
3328   {
3329     \int_case:nnF \c@jCol
3330     {
3331       0 { \@@_error:nn { in~first~col } \Iddots }
3332       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
3333     }
3334     {
3335       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3336       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
3337       { #1 , down = #2 , up = #3 }
3338     }
3339   }
3340   \bool_if:NF \l_@@_nullify_dots_bool
3341   { \phantom { \ensuremath { \@@_old_iddots } } }
3342   \bool_gset_true:N \g_@@_empty_cell_bool
3343 }
3344 }

```

End of the `\AtBeginDocument`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

3345 \keys_define:nn { NiceMatrix / Ddots }
3346 {
3347   draw-first .bool_set:N = \l_@@_draw_first_bool ,
3348   draw-first .default:n = true ,
3349   draw-first .value_forbidden:n = true
3350 }

```

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

```

3351 \cs_new_protected:Npn \@@_Hspace:
3352 {
3353   \bool_gset_true:N \g_@@_empty_cell_bool
3354   \hspace
3355 }

```

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

```

3356 \cs_set_eq:NN \@@_old_multicolumn \multicolumn
3357 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3358 {

```

We have to act in an expandable way since it will begin by a `\multicolumn`.

```

3359 \exp_args:NNe
3360 \@@_old_multicolumn
3361 { #1 }

```

We will have to replace `\tl_lower_case:n` in the future since it seems to be deprecated.

```

3362 {
3363   \exp_args:Ne \str_case:nn { \str_foldcase:n { #2 } }
3364   {
3365     l { > \@@_Cell: l < \@@_end_Cell: }
3366     r { > \@@_Cell: r < \@@_end_Cell: }
3367     c { > \@@_Cell: c < \@@_end_Cell: }
3368     { l | } { > \@@_Cell: l < \@@_end_Cell: | }
3369     { r | } { > \@@_Cell: r < \@@_end_Cell: | }
3370     { c | } { > \@@_Cell: c < \@@_end_Cell: | }

```

```

3371         { | l } { | > \@@_Cell: l < \@@_end_Cell: }
3372         { | r } { | > \@@_Cell: r < \@@_end_Cell: }
3373         { | c } { | > \@@_Cell: c < \@@_end_Cell: }
3374         { | l | } { | > \@@_Cell: l < \@@_end_Cell: | }
3375         { | r | } { | > \@@_Cell: r < \@@_end_Cell: | }
3376         { | c | } { | > \@@_Cell: c < \@@_end_Cell: | }
3377     }
3378 }
3379 { #3 }

```

The `\peek_remove_spaces:n` is mandatory.

```

3380 \peek_remove_spaces:n
3381 {
3382     \int_compare:nNnT #1 > 1
3383     {
3384         \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
3385         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3386         \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
3387         \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
3388         {
3389             { \int_use:N \c@iRow }
3390             { \int_use:N \c@jCol }
3391             { \int_use:N \c@iRow }
3392             { \int_eval:n { \c@jCol + #1 - 1 } }
3393         }
3394     }
3395     \int_gadd:Nn \c@jCol { #1 - 1 }
3396     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3397     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3398 }
3399 }

```

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`.

```

3400 \cs_new:Npn \@@_Hdotsfor:
3401 {
3402     \bool_lazy_and:nnTF
3403     { \int_compare_p:nNn \c@jCol = 0 }
3404     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
3405     {
3406         \bool_if:NTF \g_@@_after_col_zero_bool
3407         {
3408             \multicolumn { 1 } { c } { }
3409             \@@_Hdotsfor_i
3410         }
3411         { \@@_fatal:n { Hdotsfor~in~col~0 } }
3412     }
3413     {
3414         \multicolumn { 1 } { c } { }
3415         \@@_Hdotsfor_i
3416     }
3417 }

```

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:`).

```

3418 \AtBeginDocument
3419 {
3420     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3421     \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.

```

3422 \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
3423 {
3424   \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
3425   {
3426     \@@_Hdotsfor:nnnn
3427     { \int_use:N \c@iRow }
3428     { \int_use:N \c@jCol }
3429     { #2 }
3430     {
3431       #1 , #3 ,
3432       down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3433     }
3434   }
3435   \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
3436 }
3437 }

```

Enf of \AtBeginDocument.

```

3438 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
3439 {
3440   \bool_set_false:N \l_@@_initial_open_bool
3441   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

3442 \int_set:Nn \l_@@_initial_i_int { #1 }
3443 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

3444 \int_compare:nNnTF #2 = 1
3445 {
3446   \int_set:Nn \l_@@_initial_j_int 1
3447   \bool_set_true:N \l_@@_initial_open_bool
3448 }
3449 {
3450   \cs_if_exist:cTF
3451   {
3452     pgf @ sh @ ns @ \@@_env:
3453     - \int_use:N \l_@@_initial_i_int
3454     - \int_eval:n { #2 - 1 }
3455   }
3456   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
3457   {
3458     \int_set:Nn \l_@@_initial_j_int { #2 }
3459     \bool_set_true:N \l_@@_initial_open_bool
3460   }
3461 }
3462 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
3463 {
3464   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3465   \bool_set_true:N \l_@@_final_open_bool
3466 }
3467 {
3468   \cs_if_exist:cTF
3469   {
3470     pgf @ sh @ ns @ \@@_env:
3471     - \int_use:N \l_@@_final_i_int
3472     - \int_eval:n { #2 + #3 }
3473   }
3474   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
3475   {
3476     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3477     \bool_set_true:N \l_@@_final_open_bool

```



```

3478     }
3479 }
3480 \group_begin:
3481 \int_compare:nNnTF { #1 } = 0
3482 { \color { nicematrix-first-row } }
3483 {
3484     \int_compare:nNnT { #1 } = \g_@@_row_total_int
3485     { \color { nicematrix-last-row } }
3486 }
3487 \keys_set:nn { NiceMatrix / xdots } { #4 }
3488 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3489 \@@_actually_draw_Ldots:
3490 \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).

```

3491 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
3492 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
3493 }
3494 \AtBeginDocument
3495 {
3496     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3497     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3498     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
3499     {
3500         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
3501         {
3502             \@@_Vdotsfor:nnnn
3503             { \int_use:N \c@iRow }
3504             { \int_use:N \c@jCol }
3505             { #2 }
3506             {
3507                 #1 , #3 ,
3508                 down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3509             }
3510         }
3511     }
3512 }

```

Enf of `\AtBeginDocument`.

```

3513 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
3514 {
3515     \bool_set_false:N \l_@@_initial_open_bool
3516     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

3517 \int_set:Nn \l_@@_initial_j_int { #2 }
3518 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

3519 \int_compare:nNnTF #1 = 1
3520 {
3521     \int_set:Nn \l_@@_initial_i_int 1
3522     \bool_set_true:N \l_@@_initial_open_bool
3523 }
3524 {
3525     \cs_if_exist:cTF
3526     {
3527         pgf @ sh @ ns @ \@@_env:
3528         - \int_eval:n { #1 - 1 }
3529         - \int_use:N \l_@@_initial_j_int

```

```

3530     }
3531     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
3532     {
3533         \int_set:Nn \l_@@_initial_i_int { #1 }
3534         \bool_set_true:N \l_@@_initial_open_bool
3535     }
3536 }
3537 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
3538 {
3539     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3540     \bool_set_true:N \l_@@_final_open_bool
3541 }
3542 {
3543     \cs_if_exist:cTF
3544     {
3545         pgf @ sh @ ns @ \@@_env:
3546         - \int_eval:n { #1 + #3 }
3547         - \int_use:N \l_@@_final_j_int
3548     }
3549     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
3550     {
3551         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3552         \bool_set_true:N \l_@@_final_open_bool
3553     }
3554 }
3555 \group_begin:
3556 \int_compare:nNnTF { #2 } = 0
3557 { \color { nicematrix-first-col } }
3558 {
3559     \int_compare:nNnT { #2 } = \g_@@_col_total_int
3560     { \color { nicematrix-last-col } }
3561 }
3562 \keys_set:nn { NiceMatrix / xdots } { #4 }
3563 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3564 \@@_actually_draw_Vdots:
3565 \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).

```

3566     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
3567     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
3568 }

```

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

```

3569 \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).⁵³

```

3570 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
3571 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

⁵³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

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).

```

3572 \AtBeginDocument
3573 {
3574   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
3575   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3576   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
3577     {
3578       \group_begin:
3579       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
3580       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3581       \use:e
3582       {
3583         \@@_line_i:nn
3584         { \@@_double_int_eval:n #2 \q_stop }
3585         { \@@_double_int_eval:n #3 \q_stop }
3586       }
3587       \group_end:
3588     }
3589 }
3590 \cs_new_protected:Npn \@@_line_i:nn #1 #2
3591 {
3592   \bool_set_false:N \l_@@_initial_open_bool
3593   \bool_set_false:N \l_@@_final_open_bool
3594   \bool_if:nTF
3595     {
3596       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
3597       ||
3598       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
3599     }
3600     {
3601       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
3602     }
3603     { \@@_draw_line_ii:nn { #1 } { #2 } }
3604 }
3605 \AtBeginDocument
3606 {
3607   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
3608   {

```

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:`.

```

3609   \c_@@_pgfortikzpicture_tl
3610   \@@_draw_line_iii:nn { #1 } { #2 }
3611   \c_@@_endpgfortikzpicture_tl
3612 }
3613 }

```

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

```

3614 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
3615 {
3616   \pgfrememberpicturepositiononpagetrue
3617   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
3618   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3619   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3620   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
3621   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3622   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3623   \@@_draw_line:
3624 }

```

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 `\l_@@_colors_seq` will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: `[gray]{0.5}`).
- For the color whose index in `\l_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\l_@@_color_i_tl`. In that token list, the instructions will `\@@_rowcolor:n`, `\@@_columncolor:n` and `\@@_rectanglecolor:nn` (corresponding of `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor`).

`bigskip #1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_color_seq` doesn't only add a color to `\l_@@_colors_seq`: it also updates the corresponding token list `\l_@@_color_i_tl`.

```
3625 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
3626 {
```

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.

```
3627   \int_zero:N \l_tmpa_int
3628   \seq_map_indexed_inline:Nn \l_@@_colors_seq
3629     { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
3630   \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

First, the case where the color is a *new* color (not in the sequence).

```
3631   {
3632     \seq_put_right:Nn \l_@@_colors_seq { #1 }
3633     \tl_set:cn { l_@@_color _ \seq_count:N \l_@@_colors_seq _ tl } { #2 }
3634   }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
3635     { \tl_put_right:cn { l_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
3636   }
```

```
3637 \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`).

```
3638 \cs_new_protected:Npn \@@_actually_color:
3639 {
3640   \pgfpicture
3641   \pgf@relevantforpicturesizefalse
3642   \seq_map_indexed_inline:Nn \l_@@_colors_seq
3643     {
3644       \color ##2
3645       \use:c { l_@@_color _ ##1 _ tl }
3646       \pgfusepath { fill }
3647     }
3648   \endpgfpicture
3649 }
```

```

3650 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
3651 {
3652   \tl_set:Nn \l_tmpa_tl { #1 }
3653   \tl_set:Nn \l_tmpb_tl { #2 }
3654 }

```

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

```

3655 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
3656 {
3657   \tl_if_blank:nF { #2 }
3658   {
3659     \@@_add_to_colors_seq:xn
3660     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3661     { \@@_rowcolor:n { #3 } }
3662   }
3663 }
3664 \cs_new_protected:Npn \@@_rowcolor:n #1
3665 {
3666   \tl_set:Nn \l_@@_rows_tl { #1 }
3667   \tl_set:Nn \l_@@_cols_tl { - }

```

The command \@@_cartesian_path: takes in two implicit arguments: \l_@@_cols_tl and \l_@@_rows_tl.

```

3668   \@@_cartesian_path:
3669 }

```

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

```

3670 \NewDocumentCommand \@@_columncolor { 0 { } m m }
3671 {
3672   \tl_if_blank:nF { #2 }
3673   {
3674     \@@_add_to_colors_seq:xn
3675     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3676     { \@@_columncolor:n { #3 } }
3677   }
3678 }
3679 \cs_new_protected:Npn \@@_columncolor:n #1
3680 {
3681   \tl_set:Nn \l_@@_rows_tl { - }
3682   \tl_set:Nn \l_@@_cols_tl { #1 }

```

The command \@@_cartesian_path: takes in two implicit arguments: \l_@@_cols_tl and \l_@@_rows_tl.

```

3683   \@@_cartesian_path:
3684 }

```

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

```

3685 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
3686 {
3687   \tl_if_blank:nF { #2 }
3688   {
3689     \@@_add_to_colors_seq:xn
3690     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3691     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
3692   }
3693 }

```

The last argument is the radius of the corners of the rectangle.

```

3694 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
3695 {
3696   \tl_if_blank:nF { #2 }
3697   {
3698     \@@_add_to_colors_seq:xn
3699     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
3700     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
3701   }
3702 }

```

The last argument is the radius of the corners of the rectangle.

```

3703 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
3704 {
3705   \@@_cut_on_hyphen:w #1 \q_stop
3706   \tl_clear_new:N \l_tmpc_tl
3707   \tl_clear_new:N \l_tmpd_tl
3708   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
3709   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
3710   \@@_cut_on_hyphen:w #2 \q_stop
3711   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
3712   \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`.

```

3713   \@@_cartesian_path:n { #3 }
3714 }

```

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

```

3715 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
3716 {
3717   \clist_map_inline:nn { #3 }
3718   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
3719 }

```

```

3720 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
3721 {
3722   \int_step_inline:nn { \int_use:N \c@iRow }
3723   {
3724     \int_step_inline:nn { \int_use:N \c@jCol }
3725     {
3726       \int_if_even:nTF { ####1 + ##1 }
3727       { \@@_cellcolor [ #1 ] { #2 } }
3728       { \@@_cellcolor [ #1 ] { #3 } }
3729       { ##1 - ####1 }
3730     }
3731   }
3732 }

```

```

3733 \keys_define:nn { NiceMatrix / rowcolors }
3734 {
3735   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
3736   respect-blocks .default:n = true ,
3737   cols .tl_set:N = \l_@@_cols_tl ,
3738   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
3739   restart .default:n = true ,
3740   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
3741 }

```

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.

```
3742 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
3743 {
```

The group is for the options.

```
3744 \group_begin:
3745 \tl_clear_new:N \l_@@_cols_tl
3746 \tl_set:Nn \l_@@_cols_tl { - }
3747 \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.

```
3748 \bool_set_true:N \l_tmpa_bool
3749 \bool_lazy_and:nnT
3750 \l_@@_respect_blocks_bool
3751 {
3752 \cs_if_exist_p:c
3753 { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } }
3754 {
```

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`).

```
3755 \seq_set_eq:Nc \l_tmpb_seq
3756 { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
3757 \seq_set_filter:Nn \l_tmpa_seq \l_tmpb_seq
3758 { \@@_not_in_exterior_p:nnnn ##1 }
3759 }
3760 \pgfpicture
3761 \pgf@relevantforpicturesizefalse
3762 \clist_map_inline:nn { #2 }
3763 {
3764 \tl_set:Nn \l_tmpa_tl { ##1 }
3765 \tl_if_in:NnTF \l_tmpa_tl { - }
3766 { \@@_cut_on_hyphen:w ##1 \q_stop }
3767 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
3768
```

The counter `\l_tmpa_int` will be the index of the loop.

```
3768 \int_set:Nn \l_tmpa_int \l_tmpa_tl
3769 \bool_if:NTF \l_@@_rowcolors_restart_bool
3770 { \bool_set_true:N \l_tmpa_bool }
3771 { \bool_set:Nn \l_tmpa_bool { \int_if_odd_p:n { \l_tmpa_tl } } }
3772 \int_zero_new:N \l_tmpc_int
3773 \int_set:Nn \l_tmpc_int \l_tmpb_tl
3774 \int_do_until:nNnn \l_tmpa_int > \l_tmpc_int
3775 {
```

We will compute in `\l_tmpb_int` the last row of the “block”.

```
3776 \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).

```
3777 \bool_lazy_and:nnT
3778 \l_@@_respect_blocks_bool
3779 {
3780 \cs_if_exist_p:c
3781 { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
3782 }
3783 {
3784 \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
3785 { \@@_intersect_our_row_p:nnnn ####1 }
3786 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn ####1 }
```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

3787     }
3788     \tl_set:Nx \l_@@_rows_tl
3789     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
3790     \bool_if:NTF \l_tmpa_bool
3791     {
3792         \tl_if_blank:nF { #3 }
3793         {
3794             \tl_if_empty:nTF { #1 }
3795             \color
3796             { \color [ #1 ] }
3797             { #3 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

3798         \@@_cartesian_path:
3799         \pgfusepath { fill }
3800     }
3801     \bool_set_false:N \l_tmpa_bool
3802 }
3803 {
3804     \tl_if_blank:nF { #4 }
3805     {
3806         \tl_if_empty:nTF { #1 }
3807         \color
3808         { \color [ #1 ] }
3809         { #4 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

3810         \@@_cartesian_path:
3811         \pgfusepath { fill }
3812     }
3813     \bool_set_true:N \l_tmpa_bool
3814 }
3815     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
3816 }
3817 }
3818 \endpgfpicture
3819 \group_end:
3820 }

```

```

3821 \cs_new_protected:Npn \@@_rowcolors_i:nnnn #1 #2 #3 #4
3822 {
3823     \int_compare:nNnT { #3 } > \l_tmpb_int
3824     { \int_set:Nn \l_tmpb_int { #3 } }
3825 }

```

```

3826 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnn p
3827 {
3828     \bool_lazy_or:nnTF
3829     { \int_compare_p:nNn { #4 } = \c_zero_int }
3830     { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
3831     \prg_return_false:
3832     \prg_return_true:
3833 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

3834 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnn p
3835 {
3836     \bool_if:nTF

```



```

3837 {
3838   \int_compare_p:n { #1 <= \l_tmpa_int }
3839   &&
3840   \int_compare_p:n { \l_tmpa_int <= #3 }
3841 }
3842 \prg_return_true:
3843 \prg_return_false:
3844 }
3845 \cs_new:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

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.

```

3846 \cs_new_protected:Npn \@@_cartesian_path:n #1
3847 {

```

We begin the loop over the columns.

```

3848   \clist_map_inline:Nn \l_@@_cols_tl
3849   {
3850     \tl_set:Nn \l_tmpa_tl { ##1 }
3851     \tl_if_in:NnTF \l_tmpa_tl { - }
3852     { \@@_cut_on_hyphen:w ##1 \q_stop }
3853     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
3854     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
3855     \tl_if_empty:NT \l_tmpb_tl
3856     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
3857     \int_compare:nNnT \l_tmpb_tl > \c@jCol
3858     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

If we decide to provide the commands `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors` and `\chessboardcolors` available 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 rows below in the same command `\@@_cartesian_path:`.

```

3859     \@@_qpoint:n { col - \l_tmpa_tl }
3860     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
3861     { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
3862     { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
3863     \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
3864     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

3865     \clist_map_inline:Nn \l_@@_rows_tl
3866     {
3867       \tl_set:Nn \l_tmpa_tl { ####1 }
3868       \tl_if_in:NnTF \l_tmpa_tl { - }
3869       { \@@_cut_on_hyphen:w ####1 \q_stop }
3870       { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
3871       \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
3872       \tl_if_empty:NT \l_tmpb_tl
3873       { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
3874       \int_compare:nNnT \l_tmpb_tl > \c@iRow
3875       { \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`.

```

3876       \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
3877       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
3878       \@@_qpoint:n { row - \l_tmpa_tl }
3879       \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
3880       \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
3881       \pgfpathrectanglecorners
3882       { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
3883       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
3884     }

```

```

3885     }
3886 }

```

When the user uses the key `colortbl-like`, the following command will be linked to `\cellcolor` in the `tabular`.

```

3887 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
3888 {
3889     \tl_gput_right:Nx \g_nicematrix_code_before_tl
3890     {

```

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`).

```

3891         \cellcolor [ #1 ] { \exp_not:n { #2 } }
3892         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3893     }
3894 }

```

When the user uses the key `colortbll-like`, the following command will be linked to `\rowcolor` in the `tabular`.

```

3895 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
3896 {
3897     \tl_gput_right:Nx \g_nicematrix_code_before_tl
3898     {
3899         \exp_not:N \rectanglecolor [ #1 ] { \exp_not:n { #2 } }
3900         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3901         { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
3902     }
3903 }

```

```

3904 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
3905 {
3906     \int_compare:nNnT \c@iRow = 1
3907     {

```

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).

```

3908         \tl_gput_left:Nx \g_nicematrix_code_before_tl
3909         {
3910             \exp_not:N \columncolor [ #1 ]
3911             { \exp_not:n { #2 } } { \int_use:N \c@jCol }
3912         }
3913     }
3914 }

```

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`.

```

3915 \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`.

```

3916 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
3917 {
3918   \int_compare:nNnTF \l_@@_first_col_int = 0
3919     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
3920     {
3921       \int_compare:nNnTF \c@jCol = 0
3922         {
3923           \int_compare:nNnF \c@iRow = { -1 }
3924             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
3925           }
3926         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
3927       }
3928   }

```

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.

```

3929 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
3930 {
3931   \int_compare:nNnF \c@iRow = 0
3932     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
3933   }

```

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 `|`.

```

3934 \cs_new_protected:Npn \@@_vline:nn #1 #2
3935 {

```

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).

```

3936   \int_compare:nNnT { #1 } < { \c@jCol + 2 }
3937   {
3938     \pgfpicture
3939     \@@_vline_i:nn { #1 } { #2 }
3940     \endpgfpicture
3941   }
3942 }
3943 \cs_new_protected:Npn \@@_vline_i:nn #1 #2
3944 {

```

`\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`.

```

3945   \tl_set:Nx \l_tmpb_tl { #1 }
3946   \tl_clear_new:N \l_tmpc_tl
3947   \int_step_variable:nNn \c@iRow \l_tmpa_tl
3948   {

```

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.

```

3949     \bool_gset_true:N \g_tmpa_bool
3950     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
3951       { \@@_test_vline_in_block:nnnn ##1 }
3952     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
3953       { \@@_test_vline_in_block:nnnn ##1 }
3954     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq

```

```

3955     { \@@_test_vline_in_stroken_block:nnnn ##1 }
3956     \clist_if_empty:NF \l_@@_except_corners_clist
3957     \@@_test_in_corner_v:
3958     \bool_if:NTF \g_tmpa_bool
3959     {
3960         \tl_if_empty:NT \l_tmpc_tl
We keep in memory that we have a rule to draw.
3961         { \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl }
3962     }
3963     {
3964         \tl_if_empty:NF \l_tmpc_tl
3965         {
3966             \@@_vline_ii:nnnn
3967             { #1 }
3968             { #2 }
3969             \l_tmpc_tl
3970             { \int_eval:n { \l_tmpa_tl - 1 } }
3971             \tl_clear:N \l_tmpc_tl
3972         }
3973     }
3974 }
3975 \tl_if_empty:NF \l_tmpc_tl
3976 {
3977     \@@_vline_ii:nnnn
3978     { #1 }
3979     { #2 }
3980     \l_tmpc_tl
3981     { \int_use:N \c@iRow }
3982     \tl_clear:N \l_tmpc_tl
3983 }
3984 }

3985 \cs_new_protected:Npn \@@_test_in_corner_v:
3986 {
3987     \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
3988     {
3989         \seq_if_in:NxT
3990         \l_@@_empty_corner_cells_seq
3991         { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
3992         { \bool_set_false:N \g_tmpa_bool }
3993     }
3994     {
3995         \seq_if_in:NxT
3996         \l_@@_empty_corner_cells_seq
3997         { \l_tmpa_tl - \l_tmpb_tl }
3998         {
3999             \int_compare:nNnTF \l_tmpb_tl = 1
4000             { \bool_set_false:N \g_tmpa_bool }
4001             {
4002                 \seq_if_in:NxT
4003                 \l_@@_empty_corner_cells_seq
4004                 { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4005                 { \bool_set_false:N \g_tmpa_bool }
4006             }
4007         }
4008     }
4009 }

```

#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.

```

4010 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4

```

```

4011 {
4012   \pgfrememberpicturepositiononpagetrue
4013   \pgf@relevantforpicturesizefalse
4014   \@@_qpoint:n { row - #3 }
4015   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4016   \@@_qpoint:n { col - #1 }
4017   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4018   \@@_qpoint:n { row - \@@_succ:n { #4 } }
4019   \dim_set_eq:NN \l_tmpc_dim \pgf@y
4020   \bool_lazy_and:nnT
4021     { \int_compare_p:nNn { #2 } > 1 }
4022     { ! \tl_if_blank_p:V \CT@drsc@ }
4023     {
4024       \group_begin:
4025       \CT@drsc@
4026       \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4027       \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
4028       \dim_set:Nn \l_tmpd_dim
4029         { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4030       \pgfpathrectanglecorners
4031         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4032         { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4033       \pgfusepath { fill }
4034       \group_end:
4035     }
4036   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4037   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4038   \prg_replicate:nn { #2 - 1 }
4039   {
4040     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4041     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4042     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4043     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4044   }
4045   \CT@arc@
4046   \pgfsetlinewidth { 1.1 \arrayrulewidth }
4047   \pgfsetrectcap
4048   \pgfusepathqstroke
4049 }

```

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 `except-corners` is not used).

```

4050 \cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
4051 { \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

```

The command `\@@_draw_hlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `except-corners` is used).

```

4052 \cs_new_protected:Npn \@@_draw_vlines:
4053 {
4054   \int_step_inline:nnn
4055     { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
4056     { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
4057     {
4058       \tl_if_eq:NnF \l_@@_vlines_clist { all }
4059       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4060       { \@@_vline:nn { ##1 } 1 }
4061     }
4062 }

```

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`.

```

4063 \cs_new_protected:Npn \@@_hline:nn #1 #2
4064 {
4065   \pgfpicture
4066   \@@_hline_i:nn { #1 } { #2 }
4067   \endpgfpicture
4068 }
4069 \cs_new_protected:Npn \@@_hline_i:nn #1 #2
4070 {

```

`\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`.

```

4071   \tl_set:Nn \l_tmpa_tl { #1 }
4072   \tl_clear_new:N \l_tmpc_tl
4073   \int_step_variable:nNn \c@jCol \l_tmpb_tl
4074   {

```

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.

```

4075     \bool_gset_true:N \g_tmpa_bool
4076     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4077     { \@@_test_hline_in_block:nnnn ##1 }
4078     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4079     { \@@_test_hline_in_block:nnnn ##1 }
4080     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4081     { \@@_test_hline_in_stroken_block:nnnn ##1 }
4082     \clist_if_empty:NF \l_@@_except_corners_clist \@@_test_in_corner_h:
4083     \bool_if:NTF \g_tmpa_bool
4084     {
4085       \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4086       { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
4087     }
4088     {
4089       \tl_if_empty:NF \l_tmpc_tl
4090       {
4091         \@@_hline_ii:nnnn
4092         { #1 }
4093         { #2 }
4094         \l_tmpc_tl
4095         { \int_eval:n { \l_tmpb_tl - 1 } }
4096         \tl_clear:N \l_tmpc_tl
4097       }
4098     }
4099   }
4100   \tl_if_empty:NF \l_tmpc_tl
4101   {
4102     \@@_hline_ii:nnnn
4103     { #1 }
4104     { #2 }
4105     \l_tmpc_tl
4106     { \int_use:N \c@jCol }
4107     \tl_clear:N \l_tmpc_tl
4108   }
4109 }

```

```

4110 \cs_new_protected:Npn \@@_test_in_corner_h:
4111 {
4112   \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
4113   {
4114     \seq_if_in:NxT
4115     \l_@@_empty_corner_cells_seq
4116     { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4117     { \bool_set_false:N \g_tmpa_bool }
4118   }
4119   {
4120     \seq_if_in:NxT
4121     \l_@@_empty_corner_cells_seq
4122     { \l_tmpa_tl - \l_tmpb_tl }
4123     {
4124       \int_compare:nNnTF \l_tmpa_tl = 1
4125       { \bool_set_false:N \g_tmpa_bool }
4126       {
4127         \seq_if_in:NxT
4128         \l_@@_empty_corner_cells_seq
4129         { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4130         { \bool_set_false:N \g_tmpa_bool }
4131       }
4132     }
4133   }
4134 }

```

#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.

```

4135 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
4136 {
4137   \pgfrememberpicturepositiononpagetrue
4138   \pgf@relevantforpicturesizefalse
4139   \@@_qpoint:n { col - #3 }
4140   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4141   \@@_qpoint:n { row - #1 }
4142   \dim_set_eq:NN \l_tmpb_dim \pgf@y
4143   \@@_qpoint:n { col - \@@_succ:n { #4 } }
4144   \dim_set_eq:NN \l_tmpc_dim \pgf@x
4145   \bool_lazy_and:nnT
4146   { \int_compare_p:nNn { #2 } > 1 }
4147   { ! \tl_if_blank_p:V \CT@drsc@ }
4148   {
4149     \group_begin:
4150     \CT@drsc@
4151     \dim_set:Nn \l_tmpd_dim
4152     { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4153     \pgfpathrectanglecorners
4154     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4155     { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4156     \pgfusepathqfill
4157     \group_end:
4158   }
4159   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4160   \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4161   \prg_replicate:nn { #2 - 1 }
4162   {
4163     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4164     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4165     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4166     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4167   }
4168   \CT@arc@
4169   \pgfsetlinewidth { 1.1 \arrayrulewidth }

```

```

4170 \pgfsetrectcap
4171 \pgfusepathqstroke
4172 }

4173 \cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
4174 { \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as `\Cdots` and in the corners (if the key `except-corners` is used).

```

4175 \cs_new_protected:Npn \@@_draw_hlines:
4176 {
4177   \int_step_inline:nnn
4178   { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
4179   { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
4180   {
4181     \tl_if_eq:NnF \l_@@_hlines_clist { all }
4182     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
4183     { \@@_hline:nn { ##1 } 1 }
4184   }
4185 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

4186 \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.

```

4187 \cs_set:Npn \@@_Hline_i:n #1
4188 {
4189   \peek_meaning_ignore_spaces:NTF \Hline
4190   { \@@_Hline_ii:nn { #1 + 1 } }
4191   { \@@_Hline_iii:n { #1 } }
4192 }

4193 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }

4194 \cs_set:Npn \@@_Hline_iii:n #1
4195 {
4196   \skip_vertical:n
4197   {
4198     \arrayrulewidth * ( #1 )
4199     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
4200   }
4201   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4202   { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
4203   \ifnum 0 = ` { \fi }
4204 }

```

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`.

```

4205 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4
4206 {
4207   \bool_lazy_all:nT
4208   {
4209     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
4210     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4211     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4212     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }

```



```

4213     }
4214     { \bool_gset_false:N \g_tmpa_bool }
4215 }

```

The same for vertical rules.

```

4216 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4
4217 {
4218   \bool_lazy_all:nT
4219   {
4220     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4221     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4222     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
4223     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4224   }
4225   { \bool_gset_false:N \g_tmpa_bool }
4226 }

4227 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
4228 {
4229   \bool_lazy_all:nT
4230   {
4231     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4232     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
4233     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4234     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4235   }
4236   { \bool_gset_false:N \g_tmpa_bool }
4237 }

4238 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
4239 {
4240   \bool_lazy_all:nT
4241   {
4242     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4243     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4244     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4245     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
4246   }
4247   { \bool_gset_false:N \g_tmpa_bool }
4248 }

```

The key except-corners

When the key `except-corners` is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

```

4249 \cs_new_protected:Npn \@@_compute_corners:
4250 {

```

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

```

4251   \seq_clear_new:N \l_@@_empty_corner_cells_seq
4252   \clist_map_inline:Nn \l_@@_except_corners_clist
4253   {
4254     \str_case:nnF { ##1 }
4255     {
4256       { NW }
4257       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
4258       { NE }
4259       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
4260       { SW }
4261       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
4262       { SE }
4263       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }

```

```

4264     }
4265     { \@@_error:nn { bad~corner } { ##1 } }
4266 }
4267 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_empty_corner_cells_seq`.

The six arguments of `\@@_compute_a_corner:nnnnnn` are as follow:

- #1 and #2 are the number of row and column of the cell which is actually in the corner;
- #3 and #4 are the steps in rows and the step in columns when moving from the corner;
- #5 is the number of the final row when scanning the rows from the corner;
- #6 is the number of the final column when scanning the columns from the corner.

```

4268 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
4269 {

```

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.

```

4270   \bool_set_false:N \l_tmpa_bool
4271   \int_zero_new:N \l_@@_last_empty_row_int
4272   \int_set:Nn \l_@@_last_empty_row_int { #1 }
4273   \int_step_inline:nnnn { #1 } { #3 } { #5 }
4274   {
4275     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
4276     \bool_lazy_or:nnTF
4277     {
4278       \cs_if_exist_p:c
4279       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
4280     }
4281     \l_tmpb_bool
4282     { \bool_set_true:N \l_tmpa_bool }
4283     {
4284       \bool_if:NF \l_tmpa_bool
4285       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
4286     }
4287   }

```

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

```

4288   \bool_set_false:N \l_tmpa_bool
4289   \int_zero_new:N \l_@@_last_empty_column_int
4290   \int_set:Nn \l_@@_last_empty_column_int { #2 }
4291   \int_step_inline:nnnn { #2 } { #4 } { #6 }
4292   {
4293     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
4294     \bool_lazy_or:nnTF
4295     \l_tmpb_bool
4296     {
4297       \cs_if_exist_p:c
4298       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
4299     }
4300     { \bool_set_true:N \l_tmpa_bool }
4301     {
4302       \bool_if:NF \l_tmpa_bool
4303       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
4304     }
4305   }

```

Now, we loop over the rows.

```
4306 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
4307 {
```

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

```
4308 \bool_set_false:N \l_tmpa_bool
4309 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
4310 {
4311 \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
4312 \bool_lazy_or:nnTF
4313 \l_tmpb_bool
4314 {
4315 \cs_if_exist_p:c
4316 { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
4317 }
4318 { \bool_set_true:N \l_tmpa_bool }
4319 {
4320 \bool_if:NF \l_tmpa_bool
4321 {
4322 \int_set:Nn \l_@@_last_empty_column_int { ####1 }
4323 \seq_put_right:Nn
4324 \l_@@_empty_corner_cells_seq
4325 { ##1 - ####1 }
4326 }
4327 }
4328 }
4329 }
4330 }
```

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`).

```
4331 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
4332 {
4333 \int_set:Nn \l_tmpa_int { #1 }
4334 \int_set:Nn \l_tmpb_int { #2 }
4335 \bool_set_false:N \l_tmpb_bool
4336 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4337 { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
4338 }
4339 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6
4340 {
4341 \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
4342 {
4343 \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
4344 {
4345 \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
4346 {
4347 \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
4348 { \bool_set_true:N \l_tmpb_bool }
4349 }
4350 }
4351 }
4352 }
```

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`.

```

4353 \cs_new:Npn \@@_hdottedline:
4354 {
4355   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
4356   \@@_hdottedline_i:
4357 }

```

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

```

4358 \cs_new_protected:Npn \@@_hdottedline_i:
4359 {

```

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).

```

4360   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4361   { \@@_hdottedline:n { \int_use:N \c@iRow } }
4362 }

```

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.

```

4363 \AtBeginDocument
4364 {

```

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}`).

```

4365   \cs_new_protected:Npx \@@_hdottedline:n #1
4366   {
4367     \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
4368     \bool_set_true:N \exp_not:N \l_@@_final_open_bool
4369     \c_@@_pgfortikzpicture_tl
4370     \@@_hdottedline_i:n { #1 }
4371     \c_@@_endpgfortikzpicture_tl
4372   }
4373 }

```

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

```

4374 \cs_new_protected:Npn \@@_hdottedline_i:n #1
4375 {
4376   \pgfrememberpicturepositiononpagetrue
4377   \@@_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).

```

4378   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4379   \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
4380   \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}

```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

```

4381 \@@_qpoint:n { col - 1 }
4382 \dim_set:Nn \l_@@_x_initial_dim
4383 {
4384 \pgf@x +

```

We do a reduction by `\arraycolsep` for the environments with delimiters (and not for the other).

```

4385 \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4386 - \l_@@_left_margin_dim
4387 }
4388 \@@_qpoint:n { col - \@@_succ:n \c@jCol }
4389 \dim_set:Nn \l_@@_x_final_dim
4390 {
4391 \pgf@x -
4392 \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4393 + \l_@@_right_margin_dim
4394 }

```

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.

```

4395 \tl_if_eq:NnF \l_@@_left_delim_tl (
4396 { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
4397 \tl_if_eq:NnF \l_@@_right_delim_tl )
4398 { \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`.

```

4399 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4400 \@@_draw_line:
4401 }

```

Vertical dotted lines

```

4402 \cs_new_protected:Npn \@@_vdottedline:n #1
4403 {
4404 \bool_set_true:N \l_@@_initial_open_bool
4405 \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”.

```

4406 \bool_if:NTF \c_@@_tikz_loaded_bool
4407 {
4408 \tikzpicture
4409 \@@_vdottedline_i:n { #1 }
4410 \endtikzpicture
4411 }
4412 {
4413 \pgfpicture
4414 \@@_vdottedline_i:n { #1 }
4415 \endpgfpicture
4416 }
4417 }

4418 \cs_new_protected:Npn \@@_vdottedline_i:n #1
4419 {

```

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.

```
4420 \CT@arc@
4421 \pgfrememberpicturepositiononpagetrue
4422 \@@_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).

```
4423 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
4424 \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
4425 \@@_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.

```
4426 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
4427 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
4428 \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`.

```
4429 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4430 \@@_draw_line:
4431 }
```

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.

```
4432 \bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```
4433 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
4434 {
4435   auto-columns-width .code:n =
4436   {
4437     \bool_set_true:N \l_@@_block_auto_columns_width_bool
4438     \dim_gzero_new:N \g_@@_max_cell_width_dim
4439     \bool_set_true:N \l_@@_auto_columns_width_bool
4440   }
4441 }

4442 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
4443 {
4444   \int_gincr:N \g_@@_NiceMatrixBlock_int
4445   \dim_zero:N \l_@@_columns_width_dim
4446   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
4447   \bool_if:NT \l_@@_block_auto_columns_width_bool
4448   {
4449     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4450     {
4451       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
4452       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4453     }
4454   }
4455 }
```

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`).

```

4456 {
4457   \bool_if:NT \l_@@_block_auto_columns_width_bool
4458   {
4459     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
4460     \iow_shipout:Nx \@mainaux
4461     {
4462       \cs_gset:cpn
4463       { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
4464       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
4465     }
4466     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
4467   }
4468 }

```

The extra nodes

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

```

4469 \cs_generate_variant:Nn \dim_min:nn { v n }
4470 \cs_generate_variant:Nn \dim_max:nn { v n }

```

The following command is called in `\@@_use_arraybox_with_notes_c:` just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks dans that construction uses the standard medium nodes).

```

4471 \cs_new_protected:Npn \@@_create_extra_nodes:
4472 {
4473   \bool_if:nTF \l_@@_medium_nodes_bool
4474   {
4475     \bool_if:nTF \l_@@_large_nodes_bool
4476     \@@_create_medium_and_large_nodes:
4477     \@@_create_medium_nodes:
4478   }
4479   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
4480 }

```

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`.

```

4481 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
4482 {

```

```

4483 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4484 {
4485   \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
4486   \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
4487   \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
4488   \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
4489 }
4490 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4491 {
4492   \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
4493   \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
4494   \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
4495   \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
4496 }

```

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

```

4497 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4498 {
4499   \int_step_variable:nnNn
4500   \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.

```

4501 {
4502   \cs_if_exist:cT
4503   { 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`.

```

4504 {
4505   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
4506   \dim_set:cn { l_@@_row_\@@_i: _min_dim }
4507   { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
4508   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
4509   {
4510     \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
4511     { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
4512   }

```

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`.

```

4513   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
4514   \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
4515   { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
4516   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
4517   {
4518     \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
4519     { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
4520   }
4521 }
4522 }
4523 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

4524 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4525 {
4526   \dim_compare:nnT
4527   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
4528   {
4529     \@@_qpoint:n { row - \@@_i: - base }
4530     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
4531     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
4532   }
4533 }

```



```

4534 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4535 {
4536   \dim_compare:nNnT
4537     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
4538     {
4539       \@@_qpoint:n { col - \@@_j: }
4540       \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
4541       \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
4542     }
4543   }
4544 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

4545 \cs_new_protected:Npn \@@_create_medium_nodes:
4546 {
4547   \pgfpicture
4548     \pgfrememberpicturepositiononpagetrue
4549     \pgf@relevantforpicturesizefalse
4550     \@@_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”.

```

4551   \tl_set:Nn \l_@@_suffix_tl { -medium }
4552   \@@_create_nodes:
4553   \endpgfpicture
4554 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁵⁴. 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:`.

```

4555 \cs_new_protected:Npn \@@_create_large_nodes:
4556 {
4557   \pgfpicture
4558     \pgfrememberpicturepositiononpagetrue
4559     \pgf@relevantforpicturesizefalse
4560     \@@_computations_for_medium_nodes:
4561     \@@_computations_for_large_nodes:
4562     \tl_set:Nn \l_@@_suffix_tl { -large }
4563     \@@_create_nodes:
4564   \endpgfpicture
4565 }

4566 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
4567 {
4568   \pgfpicture
4569     \pgfrememberpicturepositiononpagetrue
4570     \pgf@relevantforpicturesizefalse
4571     \@@_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”.

```

4572   \tl_set:Nn \l_@@_suffix_tl { -medium }
4573   \@@_create_nodes:
4574   \@@_computations_for_large_nodes:
4575   \tl_set:Nn \l_@@_suffix_tl { -large }
4576   \@@_create_nodes:
4577   \endpgfpicture
4578 }

```

⁵⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

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.

```

4579 \cs_new_protected:Npn \@@_computations_for_large_nodes:
4580 {
4581   \int_set:Nn \l_@@_first_row_int 1
4582   \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`.

```

4583   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
4584   {
4585     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
4586     {
4587       (
4588         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
4589         \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4590       )
4591       / 2
4592     }
4593     \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
4594     { l_@@_row _ \@@_i: _ min _ dim }
4595   }
4596   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
4597   {
4598     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
4599     {
4600       (
4601         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
4602         \dim_use:c
4603         { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4604       )
4605       / 2
4606     }
4607     \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
4608     { l_@@_column _ \@@_j: _ max _ dim }
4609   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

4610   \dim_sub:cn
4611   { l_@@_column _ 1 _ min _ dim }
4612   \l_@@_left_margin_dim
4613   \dim_add:cn
4614   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
4615   \l_@@_right_margin_dim
4616 }

```

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).

```

4617 \cs_new_protected:Npn \@@_create_nodes:
4618 {
4619   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4620   {
4621     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4622     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

4623       \@@_pgf_rect_node:nnnnn
4624       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4625       { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }

```

```

4626         { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
4627         { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
4628         { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
4629     \str_if_empty:NF \l_@@_name_str
4630     {
4631         \pgfnodealias
4632         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4633         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4634     }
4635 }
4636 }

```

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 .

```

4637     \seq_mapthread_function:NNN
4638     \g_@@_multicolumn_cells_seq
4639     \g_@@_multicolumn_sizes_seq
4640     \@@_node_for_multicolumn:nn
4641 }

4642 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
4643 {
4644     \cs_set_nopar:Npn \@@_i: { #1 }
4645     \cs_set_nopar:Npn \@@_j: { #2 }
4646 }

```

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”).

```

4647 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
4648 {
4649     \@@_extract_coords_values: #1 \q_stop
4650     \@@_pgf_rect_node:nnnnn
4651     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
4652     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
4653     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
4654     { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
4655     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
4656     \str_if_empty:NF \l_@@_name_str
4657     {
4658         \pgfnodealias
4659         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
4660         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
4661     }
4662 }

```

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.

```

4663 \keys_define:nn { NiceMatrix / Block / FirstPass }
4664 {
4665     l .code:n = \tl_set:Nn \l_@@_pos_of_block_tl l ,
4666     l .value_forbidden:n = true ,
4667     r .code:n = \tl_set:Nn \l_@@_pos_of_block_tl r ,
4668     r .value_forbidden:n = true ,

```

```

4669   c .code:n = \tl_set:Nn \l_@@_pos_of_block_tl c ,
4670   c .value_forbidden:n = true ,
4671   color .tl_set:N = \l_@@_color_tl ,
4672   color .value_required:n = true ,
4673 }

```

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.

```

4674 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
4675 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not been provided by the user, you use `1-1` (that is to say a block of only one cell).

```

4676   \tl_if_blank:nTF { #2 } { \@@_Block_i 1-1 \q_stop } { \@@_Block_i #2 \q_stop }
4677   { #1 } { #3 } { #4 }
4678 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

4679 \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.

```

4680 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
4681 {

```

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).

```

4682   \bool_lazy_or:nnTF
4683     { \tl_if_blank_p:n { #1 } }
4684     { \str_if_eq_p:nn { #1 } { * } }
4685     { \int_set:Nn \l_tmpa_int { 100 } }
4686     { \int_set:Nn \l_tmpa_int { #1 } }
4687   \bool_lazy_or:nnTF
4688     { \tl_if_blank_p:n { #2 } }
4689     { \str_if_eq_p:nn { #2 } { * } }
4690     { \int_set:Nn \l_tmpb_int { 100 } }
4691     { \int_set:Nn \l_tmpb_int { #2 } }
4692   \int_compare:nNnTF \l_tmpb_int = 1
4693   {
4694     \tl_if_empty:NTF \l_@@_cell_type_tl
4695     { \tl_set:Nn \l_@@_pos_of_block_tl c }
4696     { \tl_set_eq:NN \l_@@_pos_of_block_tl \l_@@_cell_type_tl }
4697   }
4698   { \tl_set:Nn \l_@@_pos_of_block_tl c }

4699   \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
4700   \tl_set:Nx \l_tmpa_tl
4701   {
4702     { \int_use:N \c@iRow }
4703     { \int_use:N \c@jCol }
4704     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
4705     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
4706   }

```

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).

```

4707 \bool_lazy_or:nnTF
4708 { \int_compare_p:nNn { \l_tmpa_int } = 1 }
4709 { \int_compare_p:nNn { \l_tmpb_int } = 1 }
4710 { \exp_args:Nxx \@@_Block_iv:nnnnn }
4711 { \exp_args:Nxx \@@_Block_v:nnnnn }
4712 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
4713 }

```

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).

```

4714 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
4715 {
4716   \int_gincr:N \g_@@_block_box_int
4717   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
4718   {
4719     \tl_gput_right:Nx \g_@@_internal_code_after_tl
4720     {
4721       \@@_actually_diagbox:nnnnnn
4722       { \int_use:N \c@iRow }
4723       { \int_use:N \c@jCol }
4724       { \int_eval:n { \c@iRow + #1 - 1 } }
4725       { \int_eval:n { \c@jCol + #2 - 1 } }
4726       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
4727     }
4728   }
4729   \box_gclear_new:c
4730   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
4731   \hbox_gset:cn
4732   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
4733   {

```

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.

```

4734 \tl_if_empty:NTF \l_@@_color_tl
4735 { \int_compare:nNnT { #2 } = 1 \set@color }
4736 { \color { \l_@@_color_tl } }
4737 \group_begin:
4738 \cs_set:Npn \arraystretch { 1 }
4739 \dim_set_eq:NN \extrarowheight \c_zero_dim
4740 #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).

```

4741 \bool_if:NT \g_@@_rotate_bool { \tl_set:Nn \l_@@_pos_of_block_tl c }
4742 \bool_if:NTF \l_@@_NiceTabular_bool
4743 {
4744   \exp_args:Nnx \begin { tabular }
4745   { @ { } \l_@@_pos_of_block_tl @ { } }

```

```

4746         #5
4747     \end { tabular }
4748 }
4749 {
4750     \c_math_toggle_token
4751     \exp_args:Nnx \begin { array }
4752         { @ { } \l_@@_pos_of_block_tl @ { } }
4753         #5
4754     \end { array }
4755     \c_math_toggle_token
4756 }
4757 \group_end:
4758 }
4759 \bool_if:NT \g_@@_rotate_bool
4760 {
4761     \box_grotate:cn
4762     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4763     { 90 }
4764     \bool_gset_false:N \g_@@_rotate_bool
4765 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

4766     \int_compare:nNnT { #2 } = 1
4767     {
4768         \dim_gset:Nn \g_@@_blocks_wd_dim
4769         {
4770             \dim_max:nn
4771             \g_@@_blocks_wd_dim
4772             {
4773                 \box_wd:c
4774                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4775             }
4776         }
4777     }

```

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.

```

4778     \int_compare:nNnT { #1 } = 1
4779     {
4780         \dim_gset:Nn \g_@@_blocks_ht_dim
4781         {
4782             \dim_max:nn
4783             \g_@@_blocks_ht_dim
4784             {
4785                 \box_ht:c
4786                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4787             }
4788         }
4789         \dim_gset:Nn \g_@@_blocks_dp_dim
4790         {
4791             \dim_max:nn
4792             \g_@@_blocks_dp_dim
4793             {
4794                 \box_dp:c
4795                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
4796             }
4797         }
4798     }
4799     \seq_gput_right:Nx \g_@@_blocks_seq
4800     {
4801         \l_tmpa_tl

```

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in \l_@@_pos_of_block_tl. However, maybe there were

no key of the horizontal alignment and that's why we put a key corresponding to the value of `\l_@@_pos_of_block_tl`, which is fixed by the type of current column.

```

4802     { \exp_not:n { #3 } , \l_@@_pos_of_block_tl }
4803     {
4804         \box_use_drop:c
4805         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
4806     }
4807 }
4808 }

```

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.

```

4809 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
4810 {
4811     \seq_gput_right:Nx \g_@@_blocks_seq
4812     {
4813         \l_tmpa_tl
4814         { \exp_not:n { #3 } }
4815         \exp_not:n
4816         {
4817             {
4818                 \bool_if:NTF \l_@@_NiceTabular_bool
4819                 {
4820                     \group_begin:
4821                     \cs_set:Npn \arraystretch { 1 }
4822                     \dim_set_eq:NN \extrarowheight \c_zero_dim
4823                     #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).

```

4824         \bool_if:NT \g_@@_rotate_bool
4825         { \tl_set:Nn \l_@@_pos_of_block_tl c }
4826         \exp_args:Nnx \begin { tabular }
4827         { @ { } \l_@@_pos_of_block_tl @ { } }
4828         #5
4829         \end { tabular }
4830         \group_end:
4831     }
4832     {
4833         \group_begin:
4834         \cs_set:Npn \arraystretch { 1 }
4835         \dim_set_eq:NN \extrarowheight \c_zero_dim
4836         #4
4837         \bool_if:NT \g_@@_rotate_bool
4838         { \tl_set:Nn \l_@@_pos_of_block_tl c }
4839         \c_math_toggle_token
4840         \exp_args:Nnx \begin { array }
4841         { @ { } \l_@@_pos_of_block_tl @ { } } #5 \end { array }
4842         \c_math_toggle_token
4843         \group_end:
4844     }
4845 }
4846 }
4847 }
4848 }

```

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).

```

4849 \keys_define:nn { NiceMatrix / Block / SecondPass }
4850 {
4851   fill .tl_set:N = \l_@@_fill_tl ,
4852   fill .value_required:n = true ,
4853   draw .tl_set:N = \l_@@_draw_tl ,
4854   draw .default:n = default ,
4855   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
4856   rounded-corners .default:n = 4 pt ,
4857   color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
4858   color .value_required:n = true ,
4859   line-width .dim_set:N = \l_@@_line_width_dim ,
4860   line-width .value_required:n = true ,
4861   l .code:n = \tl_set:Nn \l_@@_pos_of_block_tl l ,
4862   l .value_forbidden:n = true ,
4863   r .code:n = \tl_set:Nn \l_@@_pos_of_block_tl r ,
4864   r .value_forbidden:n = true ,
4865   c .code:n = \tl_set:Nn \l_@@_pos_of_block_tl c ,
4866   c .value_forbidden:n = true ,
4867   unknown .code:n = @@_error:n { Unknown-key-for-Block }
4868 }

```

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.

```

4869 \cs_new_protected:Npn \@@_draw_blocks:
4870 {
4871   \cs_set_eq:NN \ialign \@@_old_ialign:
4872   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
4873 }
4874 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
4875 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

4876 \int_zero_new:N \l_@@_last_row_int
4877 \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.

```

4878 \int_compare:nNnTF { #3 } > { 99 }
4879 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
4880 { \int_set:Nn \l_@@_last_row_int { #3 } }
4881 \int_compare:nNnTF { #4 } > { 99 }
4882 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
4883 { \int_set:Nn \l_@@_last_col_int { #4 } }
4884 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
4885 {
4886   \int_compare:nTF
4887   { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
4888   {
4889     \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
4890     @@_msg_redirect_name:nn { Block-too-large~2 } { none }
4891     \group_begin:
4892     \globaldefs = 1
4893     @@_msg_redirect_name:nn { columns-not-used } { none }
4894     \group_end:
4895   }
4896   { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }

```



```

4897     }
4898     {
4899         \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
4900         { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
4901         { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
4902     }
4903 }
4904 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
4905 {

```

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).

```

4906     \seq_gput_left:Nn \g_@@_pos_of_blocks_seq { { #1 } { #2 } { #3 } { #4 } }

```

The group is for the keys.

```

4907     \group_begin:
4908     \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
4909     \bool_lazy_or:nnT
4910     { ! \tl_if_empty_p:N \l_@@_draw_tl }
4911     { \dim_compare_p:nNn \l_@@_line_width_dim > \c_zero_dim }
4912     {
4913         \tl_gput_right:Nx \g_nicematrix_code_after_tl
4914         {
4915             \@@_stroke_block:nnn
4916             { \exp_not:n { #5 } }
4917             { #1 - #2 }
4918             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
4919         }
4920         \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
4921         { { #1 } { #2 } { #3 } { #4 } }
4922     }
4923     \tl_if_empty:NF \l_@@_fill_tl
4924     {
4925         \tl_gput_right:Nx \g_nicematrix_code_before_tl
4926         {
4927             \exp_not:N \roundedrectanglecolor
4928             { \exp_not:V \l_@@_fill_tl }
4929             { #1 - #2 }
4930             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
4931             { \dim_use:N \l_@@_rounded_corners_dim }
4932         }
4933     }
4934     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
4935     {
4936         \tl_gput_right:Nx \g_@@_internal_code_after_tl
4937         {
4938             \@@_actually_diagbox:nnnnnn
4939             { #1 }
4940             { #2 }
4941             { \int_use:N \l_@@_last_row_int }
4942             { \int_use:N \l_@@_last_col_int }
4943             { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
4944         }
4945     }
4946     \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
4947     \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.

```
4948 \pgfpicture
4949 \pgfrememberpicturepositiononpagetrue
4950 \pgf@relevantforpicturesizefalse
4951 \@@_qpoint:n { row - #1 }
4952 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4953 \@@_qpoint:n { col - #2 }
4954 \dim_set_eq:NN \l_tmpb_dim \pgf@x
4955 \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
4956 \dim_set_eq:NN \l_tmpc_dim \pgf@y
4957 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
4958 \dim_set_eq:NN \l_tmpd_dim \pgf@x
```

We construct the node for the block with the name (#1-#2-block).

The function \@@_pgf_rect_node:nnnnn takes in as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```
4959 \begin { pgfscope }
4960 \@@_pgf_rect_node:nnnnn
4961 { \@@_env: - #1 - #2 - block }
4962 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
4963 \end { pgfscope }
```

We construct the short node.

```
4964 \dim_set_eq:NN \l_tmpb_dim \c_max_dim
4965 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4966 {
```

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.

```
4967 \cs_if_exist:cT
4968 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
4969 {
4970 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
4971 {
4972 \pgfpictureanchor { \@@_env: - ##1 - #2 } { west }
4973 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
4974 }
4975 }
4976 }
```

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.

```
4977 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
4978 {
4979 \@@_qpoint:n { col - #2 }
4980 \dim_set_eq:NN \l_tmpb_dim \pgf@x
```

```

4981     }
4982     \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
4983     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4984     {
4985         \cs_if_exist:cT
4986         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
4987         {
4988             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
4989             {
4990                 \pgfpointanchor
4991                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
4992                 { east }
4993                 \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
4994             }
4995         }
4996     }
4997     \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
4998     {
4999         \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5000         \dim_set_eq:NN \l_tmpd_dim \pgf@x
5001     }
5002     \@@_pgf_rect_node:nnnnn
5003     { \@@_env: - #1 - #2 - block - short }
5004     \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpe_dim

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@_pgf_rect_node:nnnnn` takes in as arguments the name of the node and two PGF points.

```

5005     \bool_if:NT \l_@@_medium_nodes_bool
5006     {
5007         \@@_pgf_rect_node:nnn
5008         { \@@_env: - #1 - #2 - block - medium }
5009         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
5010         {
5011             \pgfpointanchor
5012             { \@@_env:
5013               - \int_use:N \l_@@_last_row_int
5014               - \int_use:N \l_@@_last_col_int - medium
5015             }
5016             { south-east }
5017         }
5018     }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

5019     \int_compare:nNnTF { #1 } = { #3 }
5020     {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

5021         \int_compare:nNnTF { #1 } = 0
5022         { \l_@@_code_for_first_row_tl }
5023         {
5024             \int_compare:nNnT { #1 } = \l_@@_last_row_int
5025             \l_@@_code_for_last_row_tl
5026         }

```

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`.

```

5027         \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

5028         \pgfpointanchor
5029         { \@@_env: - #1 - #2 - block - short }
5030         {
5031             \str_case:Vn \l_@@_pos_of_block_tl

```

```

5032         {
5033             c { center }
5034             l { west }
5035             r { east }
5036         }
5037     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

5038     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
5039     \pgfset { inner~sep = \c_zero_dim }
5040     \pgfnode
5041     { rectangle }
5042     {
5043         \str_case:Vn \l_@@_pos_of_block_tl
5044         {
5045             c { base }
5046             l { base~west }
5047             r { base~east }
5048         }
5049     }
5050     { \box_use_drop:N \l_@@_cell_box } { } { }
5051 }

```

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`).

```

5052 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

5053     \int_compare:nNnT { #2 } = 0
5054     { \tl_set:Nn \l_@@_pos_of_block_tl r }
5055     \bool_if:nT \g_@@_last_col_found_bool
5056     {
5057         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5058         { \tl_set:Nn \l_@@_pos_of_block_tl l }
5059     }
5060     \pgftransformshift
5061     {
5062         \pgfpointanchor
5063         { \@@_env: - #1 - #2 - block - short }
5064         {
5065             \str_case:Vn \l_@@_pos_of_block_tl
5066             {
5067                 c { center }
5068                 l { west }
5069                 r { east }
5070             }
5071         }
5072     }
5073     \pgfset { inner~sep = \c_zero_dim }
5074     \pgfnode
5075     { rectangle }
5076     {
5077         \str_case:Vn \l_@@_pos_of_block_tl
5078         {
5079             c { center }
5080             l { west }
5081             r { east }
5082         }
5083     }
5084     { \box_use_drop:N \l_@@_cell_box } { } { }
5085 }
5086 \endpgfpicture
5087 \group_end:
5088 }

```

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).

```

5089 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
5090 {
5091   \group_begin:
5092   \tl_clear:N \l_@@_draw_tl
5093   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5094   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
5095   \pgfpicture
5096   \pgfrememberpicturepositiononpagetrue
5097   \pgf@relevantforpicturesizefalse
5098   \tl_if_empty:NF \l_@@_draw_tl
5099   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

5100     \str_if_eq:VnTF \l_@@_draw_tl { default }
5101     { \CT@arc@ }
5102     { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
5103   }
5104   \pgfsetcornersarced
5105   {
5106     \pgfpoint
5107     { \dim_use:N \l_@@_rounded_corners_dim }
5108     { \dim_use:N \l_@@_rounded_corners_dim }
5109   }
5110   \@@_cut_on_hyphen:w #2 \q_stop
5111   \bool_lazy_and:nnT
5112   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
5113   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
5114   {
5115     \@@_qpoint:n { row - \l_tmpa_tl }
5116     \dim_set:Nn \l_tmpb_dim { \pgf@y }
5117     \@@_qpoint:n { col - \l_tmpb_tl }
5118     \dim_set:Nn \l_tmpc_dim { \pgf@x }
5119     \@@_cut_on_hyphen:w #3 \q_stop
5120     \int_compare:nNnT \l_tmpa_tl > \c@iRow
5121     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
5122     \int_compare:nNnT \l_tmpb_tl > \c@jCol
5123     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5124     \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
5125     \dim_set:Nn \l_tmpa_dim { \pgf@y }
5126     \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
5127     \dim_set:Nn \l_tmpd_dim { \pgf@x }
5128     \pgfpathrectanglecorners
5129     { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5130     { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5131     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

5132     \pgfusepath { stroke }
5133   }
5134   \endpgfpicture
5135   \group_end:
5136 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

5137 \keys_define:nn { NiceMatrix / BlockStroke }
5138 {

```

We will uncomment the following line when we will give to the key `color` of the command `\Block` its new definition.

```

5139   color .tl_set:N = \l_@@_draw_tl ,
5140   draw .tl_set:N = \l_@@_draw_tl ,

```

```

5141 draw .default:n = default ,
5142 line-width .dim_set:N = \l_@@_line_width_dim ,
5143 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5144 rounded-corners .default:n = 4 pt
5145 }

```

How to draw the dotted lines transparently

```

5146 \cs_set_protected:Npn \@@_renew_matrix:
5147 {
5148   \RenewDocumentEnvironment { pmatrix } { } {
5149     { \pNiceMatrix }
5150     { \endpNiceMatrix }
5151   \RenewDocumentEnvironment { vmatrix } { } {
5152     { \vNiceMatrix }
5153     { \endvNiceMatrix }
5154   \RenewDocumentEnvironment { Vmatrix } { } {
5155     { \VNiceMatrix }
5156     { \endVNiceMatrix }
5157   \RenewDocumentEnvironment { bmatrix } { } {
5158     { \bNiceMatrix }
5159     { \endbNiceMatrix }
5160   \RenewDocumentEnvironment { Bmatrix } { } {
5161     { \BNiceMatrix }
5162     { \endBNiceMatrix }
5163   }

```

Automatic arrays

```

5164 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
5165 {
5166   \int_set:Nn \l_@@_nb_rows_int { #1 }
5167   \int_set:Nn \l_@@_nb_cols_int { #2 }
5168 }
5169 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
5170 {
5171   \int_zero_new:N \l_@@_nb_rows_int
5172   \int_zero_new:N \l_@@_nb_cols_int
5173   \@@_set_size:n #4 \q_stop
5174   \begin { NiceArrayWithDelims } { #1 } { #2 }
5175     { * { \l_@@_nb_cols_int } { c } } [ #3 , #5 , #7 ]
5176   \int_compare:nNnT \l_@@_first_row_int = 0
5177   {
5178     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5179     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5180     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5181   }
5182   \prg_replicate:nn \l_@@_nb_rows_int
5183   {
5184     \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`).

```

5185     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
5186     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5187   }
5188   \int_compare:nNnT \l_@@_last_row_int > { -2 }
5189   {
5190     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5191     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }

```

```

5192     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5193   }
5194   \end { NiceArrayWithDelims }
5195 }
5196 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
5197 {
5198   \cs_set_protected:cpn { #1 AutoNiceMatrix }
5199   {
5200     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
5201     \AutoNiceMatrixWithDelims { #2 } { #3 }
5202   }
5203 }
5204 \@@_define_com:nnn p ( )
5205 \@@_define_com:nnn b [ ]
5206 \@@_define_com:nnn v | |
5207 \@@_define_com:nnn V \| \|
5208 \@@_define_com:nnn B \{ \}

```

We define also an command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

5209 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
5210 {
5211   \group_begin:
5212     \bool_set_true:N \l_@@_NiceArray_bool
5213     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
5214   \group_end:
5215 }

```

The redefinition of the command `\dotfill`

```

5216 \cs_set_eq:NN \@@_old_dotfill \dotfill
5217 \cs_new_protected:Npn \@@_dotfill:
5218 {

```

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}`).

```

5219   \@@_old_dotfill
5220   \bool_if:NT \l_@@_NiceTabular_bool
5221     { \group_insert_after:N \@@_dotfill_ii: }
5222     { \group_insert_after:N \@@_dotfill_i: }
5223   }
5224 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
5225 \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`.

```

5226 \cs_new_protected:Npn \@@_dotfill_iii:
5227 { \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`.

```

5228 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
5229 {
5230   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5231   {
5232     \@@_actually_diagbox:nnnnnn
5233     { \int_use:N \c_iRow }
5234     { \int_use:N \c_jCol }
5235     { \int_use:N \c_iRow }
5236     { \int_use:N \c_jCol }
5237     { \exp_not:n { #1 } }

```

```

5238     { \exp_not:n { #2 } }
5239   }

```

We put the cell with `\diagbox` in the sequence `\g_@@_pos_of_blocks_seq` because a cell with `\diagbox` must be considered as non empty by the key `except-corners`.

```

5240   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
5241   {
5242     { \int_use:N \c@iRow }
5243     { \int_use:N \c@jCol }
5244     { \int_use:N \c@iRow }
5245     { \int_use:N \c@jCol }
5246   }
5247 }

```

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.

```

5248 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
5249 {
5250   \pgfpicture
5251   \pgf@relevantforpicturesizefalse
5252   \pgfrememberpicturepositiononpagetrue
5253   \@@_qpoint:n { row - #1 }
5254   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5255   \@@_qpoint:n { col - #2 }
5256   \dim_set_eq:NN \l_tmpb_dim \pgf@x
5257   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5258   \@@_qpoint:n { row - \@@_succ:n { #3 } }
5259   \dim_set_eq:NN \l_tmpc_dim \pgf@y
5260   \@@_qpoint:n { col - \@@_succ:n { #4 } }
5261   \dim_set_eq:NN \l_tmpd_dim \pgf@x
5262   \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
5263   {

```

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.

```

5264   \CT@arc@
5265   \pgfsetroundcap
5266   \pgfusepathqstroke
5267 }
5268 \pgfset { inner-sep = 1 pt }
5269 \pgfscope
5270 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
5271 \pgfnode { rectangle } { south-west }
5272 { \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }
5273 \endpgfscope
5274 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5275 \pgfnode { rectangle } { north-east }
5276 { \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
5277 \endpgfpicture
5278 }

```

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.

```

5279 \keys_define:nn { NiceMatrix }
5280 { CodeAfter / rules .inherit:n = NiceMatrix / rules }
5281 \keys_define:nn { NiceMatrix / CodeAfter }

```



```

5282 {
5283   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
5284   sub-matrix .value_required:n = true ,
5285   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
5286   delimiters / color .value_required:n = true ,
5287   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
5288   rules .value_required:n = true ,
5289   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
5290 }

```

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. 95.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

5291 \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`.

```

5292 \cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
5293 {
5294   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
5295   \@@_CodeAfter_ii:n
5296 }

```

We catch the argument of the command `\end` (in `#1`).

```

5297 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1
5298 {

```

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.

```

5299   \str_if_eq:eeTF \@currenenvr { #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`.

```

5300   {
5301     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
5302     \@@_CodeAfter_i:n
5303   }
5304 }

```

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 spaces between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` ou `\}`). The second argument is the number of columnn. 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.

```

5305 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
5306 {
5307   \pgfpicture
5308   \pgfrememberpicturepositiononpagetrue
5309   \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.

```

5310 \@@_qpoint:n { row - 1 }
5311 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5312 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
5313 \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).

```

5314 \bool_if:nTF { #3 }
5315 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
5316 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
5317 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5318 {
5319   \cs_if_exist:cT
5320   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5321   {
5322     \pgfpointanchor
5323     { \@@_env: - ##1 - #2 }
5324     { \bool_if:nTF { #3 } { west } { east } }
5325     \dim_set:Nn \l_tmpa_dim
5326     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
5327   }
5328 }

```

Now we can put the delimiter with a node of PGF.

```

5329 \pgfset { inner~sep = \c_zero_dim }
5330 \dim_zero:N \nulldelimiterspace
5331 \pgftransformshift
5332 {
5333   \pgfpoint
5334   { \l_tmpa_dim }
5335   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
5336 }
5337 \pgfnode
5338 { rectangle }
5339 { \bool_if:nTF { #3 } { east } { west } }
5340 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

5341 \nullfont
5342 \c_math_toggle_token
5343 \tl_if_empty:NF \l_@@_delimiters_color_tl
5344 { \color { \l_@@_delimiters_color_tl } }
5345 \bool_if:nTF { #3 } { \left #1 } { \left . }
5346 \vcenter
5347 {
5348   \nullfont
5349   \hrule \@height
5350   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
5351   \@depth \c_zero_dim
5352   \@width \c_zero_dim
5353 }
5354 \bool_if:nTF { #3 } { \right . } { \right #1 }
5355 \c_math_toggle_token
5356 }
5357 { }
5358 { }
5359 \endpgfpicture
5360 }

```

The command `\SubMatrix`

```

5361 \keys_define:nn { NiceMatrix / sub-matrix }
5362 {
5363   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
5364   extra-height .value_required:n = true ,
5365   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
5366   left-xshift .value_required:n = true ,
5367   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
5368   right-xshift .value_required:n = true ,
5369   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
5370   xshift .value_required:n = true ,
5371   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
5372   delimiters / color .value_required:n = true ,
5373   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
5374   slim .default:n = true ,
5375 }
5376 \keys_define:nn { NiceMatrix }
5377 {
5378   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
5379   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5380 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

5381 \keys_define:nn { NiceMatrix / SubMatrix }
5382 {
5383   hlines .clist_set:N = \l_@@_hlines_clist ,
5384   hlines .default:n = all ,
5385   vlines .clist_set:N = \l_@@_vlines_clist ,
5386   vlines .default:n = all ,
5387   hvlines .meta:n = { hlines, vlines } ,
5388   hvlines .value_forbidden:n = true ,
5389   name .code:n =
5390     \tl_if_empty:nTF { #1 }
5391     { \@@_error:n { Invalid-name-format } }
5392     {
5393       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
5394       {
5395         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
5396         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
5397         {
5398           \str_set:Nn \l_@@_submatrix_name_str { #1 }
5399           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
5400         }
5401       }
5402       { \@@_error:n { Invalid-name-format } }
5403     } ,
5404   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
5405   rules .value_required:n = true ,
5406   code .tl_set:N = \l_@@_code_tl ,
5407   code .value_required:n = true ,
5408   name .value_required:n = true ,
5409   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
5410 }

5411 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
5412 {
5413   \@@_cut_on_hyphen:w #3 \q_stop
5414   \tl_clear_new:N \l_tmpc_tl
5415   \tl_clear_new:N \l_tmpd_tl
5416   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5417   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl

```

```

5418 \@@_cut_on_hyphen:w #2 \q_stop
5419 \seq_gput_right:Nx \g_@@_submatrix_seq
5420 { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
5421 \tl_gput_right:Nn \g_@@_internal_code_after_tl
5422 { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
5423 }

```

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.

```

5424 \NewDocumentCommand \@@_SubMatrix { m m m m O { } }
5425 {
5426   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

5427 \tl_clear_new:N \l_@@_first_i_tl
5428 \tl_clear_new:N \l_@@_first_j_tl
5429 \tl_clear_new:N \l_@@_last_i_tl
5430 \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.

```

5431 \@@_cut_on_hyphen:w #2 \q_stop
5432 \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
5433 \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
5434 \@@_cut_on_hyphen:w #3 \q_stop
5435 \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
5436 \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
5437 \bool_lazy_or:nnTF
5438 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
5439 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
5440 { \@@_error:n { SubMatrix~too~large } }
5441 {
5442   \str_clear_new:N \l_@@_submatrix_name_str
5443   \clist_clear:N \l_@@_hlines_clist
5444   \clist_clear:N \l_@@_vlines_clist
5445   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
5446   \pgfpicture
5447   \pgfrememberpicturepositiononpagetrue
5448   \pgf@relevantforpicturesizefalse
5449   \pgfset { inner~sep = \c_zero_dim }
5450   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
5451   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currfication.

```

5452 \bool_if:NTF \l_@@_submatrix_slim_bool
5453 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
5454 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
5455 {
5456   \cs_if_exist:cT
5457   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
5458   {
5459     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
5460     \dim_set:Nn \l_@@_x_initial_dim
5461     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
5462   }

```

```

5463 \cs_if_exist:cT
5464 { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_last_j_tl }
5465 {
5466 \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_last_j_tl } { east }
5467 \dim_set:Nn \l_@@_x_final_dim
5468 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
5469 }
5470 }
5471 \l_@@_qpoint:n { row - \l_@@_first_i_tl - base }
5472 \dim_set:Nn \l_@@_y_initial_dim
5473 { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
5474 \l_@@_qpoint:n { row - \l_@@_last_i_tl - base }
5475 \dim_set:Nn \l_@@_y_final_dim
5476 { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
5477 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
5478 {
5479 \cs_if_exist:cT
5480 { pgf @ sh @ ns @ \l_@@_env: - \l_@@_first_i_tl - ##1 }
5481 {
5482 \pgfpointanchor { \l_@@_env: - \l_@@_first_i_tl - ##1 } { north }
5483 \dim_set:Nn \l_@@_y_initial_dim
5484 { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
5485 }
5486 \cs_if_exist:cT
5487 { pgf @ sh @ ns @ \l_@@_env: - \l_@@_last_i_tl - ##1 }
5488 {
5489 \pgfpointanchor { \l_@@_env: - \l_@@_last_i_tl - ##1 } { south }
5490 \dim_set:Nn \l_@@_y_final_dim
5491 { \dim_min:nn \l_@@_y_final_dim \pgf@y }
5492 }
5493 }
5494 \dim_set:Nn \l_tmpa_dim
5495 {
5496 \l_@@_y_initial_dim - \l_@@_y_final_dim +
5497 \l_@@_submatrix_extra_height_dim - \arrayrulewidth
5498 }
5499 \dim_set_eq:NN \nulldelimiterspace \c_zero_dim

```

We will draw the rules in the `\SubMatrix`.

```

5500 \group_begin:
5501 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5502 \tl_if_empty:NF \l_@@_rules_color_tl
5503 { \exp_after:wN \l_@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
5504 \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`.

```

5505 \seq_map_inline:Nn \g_@@_cols_vlism_seq
5506 {
5507 \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
5508 {
5509 \int_compare:nNnT
5510 { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
5511 {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

5512 \l_@@_qpoint:n { col - ##1 }
5513 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
5514 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
5515 \pgfusepathqstroke
5516 }
5517 }
5518 }

```

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.

```

5519 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
5520 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
5521 { \clist_map_inline:Nn \l_@@_vlines_clist }
5522 {
5523   \bool_lazy_and:nnTF
5524   { \int_compare_p:nNn { ##1 } > 0 }
5525   {
5526     \int_compare_p:nNn
5527     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
5528   {
5529     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
5530     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
5531     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
5532     \pgfusepathqstroke
5533   }
5534   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
5535 }

```

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.

```

5536 \tl_if_eq:NnTF \l_@@_hlines_clist { all }
5537 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
5538 { \clist_map_inline:Nn \l_@@_hlines_clist }
5539 {
5540   \bool_lazy_and:nnTF
5541   { \int_compare_p:nNn { ##1 } > 0 }
5542   {
5543     \int_compare_p:nNn
5544     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
5545   {
5546     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

5547 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

5548 \dim_set:Nn \l_tmpa_dim
5549 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
5550 \str_case:nn { #1 }
5551 {
5552   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
5553   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
5554   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
5555   }
5556   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

5557 \dim_set:Nn \l_tmpb_dim
5558 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
5559 \str_case:nn { #4 }
5560 {
5561   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
5562   ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
5563   \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
5564 }
5565 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5566 \pgfusepathqstroke
5567 \group_end:
5568 }
5569 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
5570 }

```

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 take into account the delimiters that we will put after).

```

5571 \str_if_empty:NF \l_@@_submatrix_name_str
5572 {
5573   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
5574   \l_@@_x_initial_dim \l_@@_y_initial_dim
5575   \l_@@_x_final_dim \l_@@_y_final_dim
5576 }
5577 \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`.

```

5578 \begin { pgfscope }
5579 \pgftransformshift
5580 {
5581   \pgfpoint
5582   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
5583   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
5584 }
5585 \str_if_empty:NTF \l_@@_submatrix_name_str
5586 { \@@_node_left:nn #1 { } }
5587 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
5588 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

5589 \pgftransformshift
5590 {
5591   \pgfpoint
5592   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
5593   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
5594 }
5595 \str_if_empty:NTF \l_@@_submatrix_name_str
5596 { \@@_node_right:nn #4 { } }
5597 {
5598   \@@_node_right:nn #4 { \@@_env: - \l_@@_submatrix_name_str - right }
5599 }
5600 \endpgfpicture
5601 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
5602 \flag_clear_new:n { nicematrix }
5603 \l_@@_code_tl
5604 }
5605 \group_end:
5606 }

```

The group was a group for the whole `\SubMatrix`.

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the *i* and *j* in specifications of nodes of the forms *i-j*, *row-i*, *col-j* and *i-lj* 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`.

```

5607 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpntanchor

```

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.

```

5608 \cs_new_protected:Npn \@@_pgfpntanchor:n #1
5609 {
5610   \use:e
5611   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpntanchor_i:nn #1 } }
5612 }

```

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 -.

```
5613 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
5614 { #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.

```
5615 \tl_const:Nn \c_@@_integers_alist_tl
5616 {
5617   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
5618   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
5619   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
5620   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
5621 }
```

```
5622 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
5623 {
```

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`.

```
5624   \tl_if_empty:nTF { #2 }
5625   {
5626     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
5627     {
5628       \flag_raise:n { nicematrix }
5629       \int_if_even:nTF { \flag_height:n { nicematrix } }
5630       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
5631       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
5632     }
5633     { #1 }
5634   }
```

If there is an hyphen, we have to see whether we have a node of the form i - j , row- i or col- j .

```
5635   { \@@_pgfpointanchor_iii:w { #1 } #2 }
5636 }
```

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`).

```
5637 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
5638 {
5639   \str_case:nnF { #1 }
5640   {
5641     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
5642     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
5643   }
```

Now the case of a node of the form i - j .

```
5644   {
5645     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
5646     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
5647   }
5648 }
```


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`).

```

5649 \cs_new_protected:Npn \@@_node_left:nn #1 #2
5650 {
5651   \pgfnode
5652     { rectangle }
5653     { east }
5654     {
5655       \nullfont
5656       \c_math_toggle_token
5657       \tl_if_empty:NF \l_@@_delimiters_color_tl
5658       { \color { \l_@@_delimiters_color_tl } }
5659       \left #1
5660       \vcenter
5661       {
5662         \nullfont
5663         \hrule \@height \l_tmpa_dim
5664         \@depth \c_zero_dim
5665         \@width \c_zero_dim
5666       }
5667       \right .
5668       \c_math_toggle_token
5669     }
5670     { #2 }
5671     { }
5672 }

```

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`).

```

5673 \cs_new_protected:Npn \@@_node_right:nn #1 #2
5674 {
5675   \pgfnode
5676     { rectangle }
5677     { west }
5678     {
5679       \nullfont
5680       \c_math_toggle_token
5681       \tl_if_empty:NF \l_@@_delimiters_color_tl
5682       { \color { \l_@@_delimiters_color_tl } }
5683       \left .
5684       \vcenter
5685       {
5686         \nullfont
5687         \hrule \@height \l_tmpa_dim
5688         \@depth \c_zero_dim
5689         \@width \c_zero_dim
5690       }
5691       \right #1
5692       \c_math_toggle_token
5693     }
5694     { #2 }
5695     { }
5696 }

```

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.

```
5697 \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.

```
5698 \bool_new:N \c_@@_footnote_bool
```

```
5699 \@@_msg_new:nnn { Unknown~option~for~package }
```

```
5700 {
5701   The~key~'\l_keys_key_str'~is~unknown.  \
5702   If~you~go~on,~it~will~be~ignored.  \
5703   For~a~list~of~the~available~keys,~type~H~<return>.
5704 }
5705 {
```

```
5706   The~available~keys~are~(in~alphabetic~order):~
5707   define-L-C-R,~
5708   footnote,~
5709   footnotehyper,~
5710   renew-dots,~and
5711   renew-matrix.
5712 }
```

```
5713 \@@_msg_new:nn { Key~transparent }
```

```
5714 {
5715   The~key~'transparent'~is~now~obsolete~(because~it's~name~
5716   is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
5717   and~'renew-matrix'.~However,~you~can~go~on.
5718 }
```

```
5719 \keys_define:nn { NiceMatrix / Package }
```

```
5720 {
5721   define-L-C-R .bool_set:N = \c_@@_define_L_C_R_bool ,
5722   define-L-C-R .default:n = true ,
5723   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
5724   renew-dots .value_forbidden:n = true ,
5725   renew-matrix .code:n = \@@_renew_matrix: ,
5726   renew-matrix .value_forbidden:n = true ,
5727   transparent .code:n =
5728     {
5729       \@@_renew_matrix:
5730       \bool_set_true:N \l_@@_renew_dots_bool
5731       \@@_error:n { Key~transparent }
5732     } ,
5733   transparent .value_forbidden:n = true,
5734   footnote .bool_set:N = \c_@@_footnote_bool ,
5735   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
5736   unknown .code:n = \@@_error:n { Unknown~option~for~package }
5737 }
```

```
5738 \ProcessKeysOptions { NiceMatrix / Package }
```

```
5739 \@@_msg_new:nn { footnote~with~footnotehyper~package }
```

```
5740 {
5741   You~can't~use~the~option~'footnote'~because~the~package~
5742   footnotehyper~has~already~been~loaded.~
5743   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
5744   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
5745   of~the~package~footnotehyper.\
5746   If~you~go~on,~the~package~footnote~won't~be~loaded.
5747 }
```

```

5748 \@@_msg_new:nn { footnotehyper~with~footnote~package }
5749 {
5750   You~can't~use~the~option~'footnotehyper'~because~the~package~
5751   footnote~has~already~been~loaded.~
5752   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
5753   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
5754   of~the~package~footnote.\\
5755   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
5756 }

```

```

5757 \bool_if:NT \c_@@_footnote_bool
5758 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

5759   \@ifclassloaded { beamer }
5760   { \bool_set_false:N \c_@@_footnote_bool }
5761   {
5762     \@ifpackageloaded { footnotehyper }
5763     { \@@_error:n { footnote~with~footnotehyper~package } }
5764     { \usepackage { footnote } }
5765   }
5766 }

```

```

5767 \bool_if:NT \c_@@_footnotehyper_bool
5768 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

5769   \@ifclassloaded { beamer }
5770   { \bool_set_false:N \c_@@_footnote_bool }
5771   {
5772     \@ifpackageloaded { footnote }
5773     { \@@_error:n { footnotehyper~with~footnote~package } }
5774     { \usepackage { footnotehyper } }
5775   }
5776   \bool_set_true:N \c_@@_footnote_bool
5777 }

```

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

```

5778 \seq_new:N \c_@@_types_of_matrix_seq
5779 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
5780 {
5781   NiceMatrix ,
5782   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
5783 }
5784 \seq_set_map_x:Nn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
5785 { \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`.

```

5786 \cs_new_protected:Npn \@@_error_too_much_cols:
5787 {
5788   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
5789   {
5790     \int_compare:nNnTF \l_@@_last_col_int = { -2 }

```

```

5791     { \@@_fatal:n { too-much-cols-for-matrix } }
5792     {
5793       \bool_if:NF \l_@@_last_col_without_value_bool
5794       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
5795     }
5796   }
5797   { \@@_fatal:n { too-much-cols-for-array } }
5798 }

```

The following command must *not* be protected since it's used in an error message.

```

5799 \cs_new:Npn \@@_message_hdotsfor:
5800 {
5801   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
5802   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is-incorrect.}
5803 }
5804 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
5805 {
5806   You-try~to-use-more-columns~than~allowed~by~your~
5807   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number~of~
5808   columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
5809   exterior~columns).~This~error~is~fatal.
5810 }
5811 \@@_msg_new:nn { too-much-cols-for-matrix }
5812 {
5813   You-try~to-use-more-columns~than~allowed~by~your~
5814   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
5815   number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
5816   'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
5817   This~error~is~fatal.
5818 }

```

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`.

```

5819 \@@_msg_new:nn { too-much-cols-for-array }
5820 {
5821   You-try~to-use-more-columns~than~allowed~by~your~
5822   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number~of~columns~is~
5823   \int_use:N \g_@@_static_num_of_col_int\
5824   ~~~~~(plus~the~potential~exterior~ones).~
5825   This~error~is~fatal.
5826 }
5827 \@@_msg_new:nn { last-col-not-used }
5828 {
5829   The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
5830   in~your~\@@_full_name_env:~.~However,~you~can~go~on.
5831 }
5832 \@@_msg_new:nn { columns-not-used }
5833 {
5834   The-preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
5835   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\
5836   You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
5837 }
5838 \@@_msg_new:nn { in-first-col }
5839 {
5840   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\
5841   If~you~go~on,~this~command~will~be~ignored.
5842 }
5843 \@@_msg_new:nn { in-last-col }
5844 {
5845   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\
5846   If~you~go~on,~this~command~will~be~ignored.
5847 }

```

```

5848 \@@_msg_new:nn { in-first-row }
5849 {
5850     You~can't~use~the~command~#1~in~the~first~row~(number~0)~of~the~array.\\
5851     If~you~go~on,~this~command~will~be~ignored.
5852 }
5853 \@@_msg_new:nn { in-last-row }
5854 {
5855     You~can't~use~the~command~#1~in~the~last~row~(exterior)~of~the~array.\\
5856     If~you~go~on,~this~command~will~be~ignored.
5857 }
5858 \@@_msg_new:nn { bad-option-for-line-style }
5859 {
5860     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
5861     is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
5862 }
5863 \@@_msg_new:nn { Unknown-key-for-xdots }
5864 {
5865     As~for~now,~there~is~only~three~key~available~here:~'color',~'line-style'~
5866     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
5867     this~key~will~be~ignored.
5868 }
5869 \@@_msg_new:nn { Unknown-key-for-rowcolors }
5870 {
5871     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
5872     (and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
5873     this~key~will~be~ignored.
5874 }
5875 \@@_msg_new:nn { ampersand-in-light-syntax }
5876 {
5877     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
5878     ~you~have~used~the~key~'light-syntax'.~This~error~is~fatal.
5879 }
5880 \@@_msg_new:nn { SubMatrix-too-large }
5881 {
5882     Your~command~\token_to_str:N \SubMatrix\
5883     can't~be~drawn~because~your~matrix~is~too~small.\\
5884     If~you~go~on,~this~command~will~be~ignored.
5885 }
5886 \@@_msg_new:nn { double-backslash-in-light-syntax }
5887 {
5888     You~can't~use~\token_to_str:N \\~to~separate~rows~because~you~have~used~
5889     the~key~'light-syntax'.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
5890     (set~by~the~key~'end-of-row').~This~error~is~fatal.
5891 }
5892 \@@_msg_new:nn { standard-cline-in-document }
5893 {
5894     The~key~'standard-cline'~is~available~only~in~the~preamble.\\
5895     If~you~go~on~this~command~will~be~ignored.
5896 }
5897 \@@_msg_new:nn { old-column-type }
5898 {
5899     The~column~type~'#1'~is~no~longer~defined~in~'nicematrix'.~
5900     Since~version~5.0,~you~have~to~use~'l',~'c'~and~'r'~instead~of~'L',~
5901     'C'~and~'R'.~You~can~also~use~the~key~'define-L-C-R'.\\
5902     This~error~is~fatal.
5903 }
5904 \@@_msg_new:nn { bad-value-for-baseline }
5905 {
5906     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~

```

```

5907     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
5908     \int_use:N \g_@@_row_total_int\ or-equal-to~'t',~'c'~or~'b'.\\
5909     If-you-go-on,~a-value-of-1~will-be-used.
5910 }
5911 \@@_msg_new:nn { Invalid-name-format }
5912 {
5913     You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
5914     \SubMatrix.\\
5915     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
5916     If-you-go-on,~this-key-will-be-ignored.
5917 }
5918 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
5919 {
5920     You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
5921     \token_to_str:N \SubMatrix\ of-your~\@@_full_name_env:\ but-that~
5922     number-is-not-valid.~If-you-go-on,~it-will-be-ignored.
5923 }
5924 \@@_msg_new:nn { empty-environment }
5925 { Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal. }
5926 \@@_msg_new:nn { Delimiter-with-small }
5927 {
5928     You-can't-put-a-delimiter-in-the-preamble-of-your~\@@_full_name_env:\
5929     because-the-key~'small'~is-in-force.\\
5930     This-error-is-fatal.
5931 }
5932 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
5933 {
5934     Your-command~\token_to_str:N\line\{#1\}\{#2\}~in-the~'code-after'~
5935     can't-be-executed-because-a-cell-doesn't-exist.\\
5936     If-you-go-on~this-command-will-be-ignored.
5937 }
5938 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
5939 {
5940     The-name~'#1'~is-already-used-for-a~\token_to_str:N \SubMatrix\
5941     in-this~\@@_full_name_env:.\\
5942     If-you-go-on,~this-key-will-be-ignored.\\
5943     For-a-list-of-the-names-already-used,~type-H<return>.
5944 }
5945 {
5946     The-names-already-defined-in-this~\@@_full_name_env:\ are:~
5947     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
5948 }
5949 \@@_msg_new:nn { r-or-l-with-preamble }
5950 {
5951     You-can't-use-the-key~'\l_keys_key_str'~in-your~\@@_full_name_env:~
5952     You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
5953     your~\@@_full_name_env:.\\
5954     If-you-go-on,~this-key-will-be-ignored.
5955 }
5956 \@@_msg_new:nn { Hdotsfor-in-col-0 }
5957 {
5958     You-can't-use~\token_to_str:N \Hdotsfor\ in-an-exterior-column-of~
5959     the-array.~This-error-is-fatal.
5960 }
5961 \@@_msg_new:nn { bad-corner }
5962 {
5963     #1-is-an-incorrect-specification-for-a-corner~(in-the-keys~
5964     'except-corners'~and~'hvlines-except-corners').~The-available~
5965     values-are:~NW,~SW,~NE~and~SE.\\
5966     If-you-go-on,~this-specification-of-corner-will-be-ignored.

```

```

5967     }
5968 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
5969 {
5970     In~the~\@@_full_name_env:,~you~must~use~the~key~
5971     'last-col'~without~value.\\
5972     However,~you~can~go~on~for~this~time~
5973     (the~value~'\l_keys_value_tl'~will~be~ignored).
5974 }
5975 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
5976 {
5977     In~\NiceMatrixoptions,~you~must~use~the~key~
5978     'last-col'~without~value.\\
5979     However,~you~can~go~on~for~this~time~
5980     (the~value~'\l_keys_value_tl'~will~be~ignored).
5981 }
5982 \@@_msg_new:nn { Block~too~large~1 }
5983 {
5984     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
5985     too~small~for~that~block. \\
5986 }
5987 \@@_msg_new:nn { Block~too~large~2 }
5988 {
5989     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
5990     \g_@@_static_num_of_col_int\
5991     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
5992     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
5993     (&)~at~the~end~of~the~first~row~of~your~
5994     \@@_full_name_env:\\.
5995     If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
5996 }
5997 \@@_msg_new:nn { unknown~column~type }
5998 {
5999     The~column~type~'#1'~in~your~\@@_full_name_env:\
6000     is~unknown. \\
6001     This~error~is~fatal.
6002 }
6003 \@@_msg_new:nn { tabularnote~forbidden }
6004 {
6005     You~can't~use~the~command~\token_to_str:N\tabularnote\
6006     ~in~a~\@@_full_name_env:~.~This~command~is~available~only~in~
6007     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
6008     If~you~go~on,~this~command~will~be~ignored.
6009 }
6010 \@@_msg_new:nn { bottomrule~without~booktabs }
6011 {
6012     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
6013     loaded~'booktabs'.\\
6014     If~you~go~on,~this~key~will~be~ignored.
6015 }
6016 \@@_msg_new:nn { enumitem~not~loaded }
6017 {
6018     You~can't~use~the~command~\token_to_str:N\tabularnote\
6019     ~because~you~haven't~loaded~'enumitem'.\\
6020     If~you~go~on,~this~command~will~be~ignored.
6021 }
6022 \@@_msg_new:nn { Wrong~last~row }
6023 {
6024     You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
6025     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
6026     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~

```

```

6027     last~row~.~You~can~avoid~this~problem~by~using~'last~row'~
6028     without~value~(more~compilations~might~be~necessary).
6029 }

6030 \@@_msg_new:nn { Yet~in~env }
6031 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }

6032 \@@_msg_new:nn { Outside~math~mode }
6033 {
6034     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
6035     (and~not~in~\token_to_str:N \vcenter).\\
6036     This~error~is~fatal.
6037 }

6038 \@@_msg_new:nn { One~letter~allowed }
6039 {
6040     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
6041     If~you~go~on,~it~will~be~ignored.
6042 }

6043 \@@_msg_new:nnn { Unknown~key~for~Block }
6044 {
6045     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
6046     \Block.\\ If~you~go~on,~it~will~be~ignored. \\
6047     For~a~list~of~the~available~keys,~type~H~<return>.
6048 }
6049 {
6050     The~available~keys~are~(in~alphabetic~order):~c,~draw,~fill,~l,~
6051     line~width,~rounded~corners~and~r.
6052 }

6053 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
6054 {
6055     The~key~'\l_keys_key_str'~is~unknown.\\
6056     If~you~go~on,~it~will~be~ignored. \\
6057     For~a~list~of~the~available~keys~in~\token_to_str:N
6058     \CodeAfter,~type~H~<return>.
6059 }
6060 {
6061     The~available~keys~are~(in~alphabetic~order):~
6062     delimiters/color,~
6063     rules~(with~the~subkeys~'color'~and~'width'),~
6064     sub~matrix~(several~subkeys)~
6065     and~xdots~(several~subkeys).~
6066     The~latter~is~for~the~command~\token_to_str:N \line.
6067 }

6068 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
6069 {
6070     The~key~'\l_keys_key_str'~is~unknown.\\
6071     If~you~go~on,~this~key~will~be~ignored. \\
6072     For~a~list~of~the~available~keys~in~\token_to_str:N
6073     \SubMatrix,~type~H~<return>.
6074 }
6075 {
6076     The~available~keys~are~(in~alphabetic~order):~
6077     'delimiters/color',~
6078     'extra~height',~
6079     'hlines',~
6080     'hvlines',~
6081     'left~xshift',~
6082     'name',~
6083     'right~xshift',~
6084     'rules'~(with~the~subkeys~'color'~and~'width'),~
6085     'slim',~
6086     'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
6087     and~'right~xshift').\\

```



```

6088     }
6089 \@@_msg_new:nnn { Unknown~key~for~notes }
6090 {
6091     The~key~'\l_keys_key_str'~is~unknown.\\
6092     If~you~go~on,~it~will~be~ignored. \\
6093     For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
6094 }
6095 {
6096     The~available~keys~are~(in~alphabetic~order):~
6097     bottomrule,~
6098     code~after,~
6099     code~before,~
6100     enumitem~keys,~
6101     enumitem~keys~para,~
6102     para,~
6103     label~in~list,~
6104     label~in~tabular~and~
6105     style.
6106 }
6107 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
6108 {
6109     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
6110     \token_to_str:N \NiceMatrixOptions. \\
6111     If~you~go~on,~it~will~be~ignored. \\
6112     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6113 }
6114 {
6115     The~available~keys~are~(in~alphabetic~order):~
6116     allow~duplicate~names,~
6117     cell~space~bottom~limit,~
6118     cell~space~limits,~
6119     cell~space~top~limit,~
6120     code~for~first~col,~
6121     code~for~first~row,~
6122     code~for~last~col,~
6123     code~for~last~row,~
6124     create~extra~nodes,~
6125     create~medium~nodes,~
6126     create~large~nodes,~
6127     delimiters/color,~
6128     end~of~row,~
6129     first~col,~
6130     first~row,~
6131     hlines,~
6132     hvlines,~
6133     hvlines~except~corners,~
6134     last~col,~
6135     last~row,~
6136     left~margin,~
6137     letter~for~dotted~lines,~
6138     light~syntax,~
6139     notes~(several~subkeys),~
6140     nullify~dots,~
6141     renew~dots,~
6142     renew~matrix,~
6143     right~margin,~
6144     rules~(with~the~subkeys~'color'~and~'width'),~
6145     small,~
6146     sub~matrix~(several~subkeys),
6147     vlines,~
6148     xdots~(several~subkeys).
6149 }
6150 \@@_msg_new:nnn { Unknown~option~for~NiceArray }

```

```

6151 {
6152   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
6153   \{NiceArray\}. \\
6154   If~you~go~on,~it~will~be~ignored. \\
6155   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6156 }
6157 {
6158   The~available~keys~are~(in~alphabetic~order):~
6159   b,~
6160   baseline,~
6161   c,~
6162   cell-space-bottom-limit,~
6163   cell-space-limits,~
6164   cell-space-top-limit,~
6165   code-after,~
6166   code-for-first-col,~
6167   code-for-first-row,~
6168   code-for-last-col,~
6169   code-for-last-row,~
6170   colortbl-like,~
6171   columns-width,~
6172   create-extra-nodes,~
6173   create-medium-nodes,~
6174   create-large-nodes,~
6175   delimiters/color,~
6176   extra-left-margin,~
6177   extra-right-margin,~
6178   first-col,~
6179   first-row,~
6180   hlines,~
6181   hvlines,~
6182   hvlines-except-corners,~
6183   last-col,~
6184   last-row,~
6185   left-margin,~
6186   light-syntax,~
6187   name,~
6188   notes/bottomrule,~
6189   notes/para,~
6190   nullify-dots,~
6191   renew-dots,~
6192   right-margin,~
6193   rules~(with~the~subkeys~'color'~and~'width'),~
6194   small,~
6195   t,~
6196   vlines,~
6197   xdots/color,~
6198   xdots/shorten~and~
6199   xdots/line-style.
6200 }

```

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).

```

6201 \@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
6202 {
6203   The~key~'\l_keys_key_str'~is~unknown~for~the~
6204   \@@_full_name_env:. \\
6205   If~you~go~on,~it~will~be~ignored. \\
6206   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6207 }
6208 {
6209   The~available~keys~are~(in~alphabetic~order):~
6210   b,~
6211   baseline,~

```

```

6212 c,~
6213 cell-space-bottom-limit,~
6214 cell-space-limits,~
6215 cell-space-top-limit,~
6216 code-after,~
6217 code-for-first-col,~
6218 code-for-first-row,~
6219 code-for-last-col,~
6220 code-for-last-row,~
6221 colortbl-like,~
6222 columns-width,~
6223 create-extra-nodes,~
6224 create-medium-nodes,~
6225 create-large-nodes,~
6226 delimiters/color,~
6227 extra-left-margin,~
6228 extra-right-margin,~
6229 first-col,~
6230 first-row,~
6231 hlines,~
6232 hvlines,~
6233 hvlines-except-corners,~
6234 l,~
6235 last-col,~
6236 last-row,~
6237 left-margin,~
6238 light-syntax,~
6239 name,~
6240 nullify-dots,~
6241 r,~
6242 renew-dots,~
6243 right-margin,~
6244 rules~(with~the~subkeys~'color'~and~'width'),~
6245 small,~
6246 t,~
6247 vlines,~
6248 xdots/color,~
6249 xdots/shorten~and~
6250 xdots/line-style.
6251 }
6252 \@@_msg_new:nnn { Unknown~option~for~NiceTabular }
6253 {
6254   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
6255   \{NiceTabular\}. \\
6256   If~you~go~on,~it~will~be~ignored. \\
6257   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6258 }
6259 {
6260   The~available~keys~are~(in~alphabetic~order):~
6261   b,~
6262   baseline,~
6263   c,~
6264   cell-space-bottom-limit,~
6265   cell-space-limits,~
6266   cell-space-top-limit,~
6267   code-after,~
6268   code-for-first-col,~
6269   code-for-first-row,~
6270   code-for-last-col,~
6271   code-for-last-row,~
6272   colortbl-like,~
6273   columns-width,~
6274   create-extra-nodes,~

```

```

6275     create-medium-nodes,~
6276     create-large-nodes,~
6277     extra-left-margin,~
6278     extra-right-margin,~
6279     first-col,~
6280     first-row,~
6281     hlines,~
6282     hvlines,~
6283     hvlines-except-corners,~
6284     last-col,~
6285     last-row,~
6286     left-margin,~
6287     light-syntax,~
6288     name,~
6289     notes/bottomrule,~
6290     notes/para,~
6291     nullify-dots,~
6292     renew-dots,~
6293     right-margin,~
6294     rules~(with~the~subkeys~'color'~and~'width'),~
6295     t,~
6296     vlines,~
6297     xdots/color,~
6298     xdots/shorten~and~
6299     xdots/line-style.
6300 }

6301 \@@_msg_new:nnn { Duplicate-name }
6302 {
6303     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
6304     the~same~environment~name~twice.~You~can~go~on,~but,~
6305     maybe,~you~will~have~incorrect~results~especially~
6306     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
6307     message~again,~use~the~key~'allow-duplicate-names'~in~
6308     '\token_to_str:N \NiceMatrixOptions'.\\
6309     For~a~list~of~the~names~already~used,~type~H~<return>. \\
6310 }
6311 {
6312     The~names~already~defined~in~this~document~are:~
6313     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
6314 }

6315 \@@_msg_new:nn { Option~auto~for~columns~width }
6316 {
6317     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
6318     If~you~go~on,~the~key~will~be~ignored.
6319 }

```

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 StackExchange⁵⁵, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁵⁶

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`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

⁵⁵cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁵⁶Before this version, there was an error when using `nicematrix` with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by `nicematrix` because they use the options `overlay` and `remember picture`.

Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`).

Possibility to draw vertical dotted lines to separate columns with the specifier `“:”` in the preamble (similar to the classical specifier `“|”` and the specifier `“:”` of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier `“:”` in the preamble.

Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.

Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of `“|”`) as `\hdotsfor` does.

Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.

Error message when the user gives an incorrect value for `last-row`.

A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol `“:”` (in the preamble of the array) and `\line` in `code-after`).

The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.

The vertical rules in the matrices (drawn by `“|”`) are now compatible with the color fixed by `colortbl`.

Correction of a bug: it was not possible to use the colon `“:”` in the preamble of an array when `pdflatex` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

The option `columns-width=auto` doesn't need any more a second compilation.

The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).

The previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange⁵⁷, 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.

⁵⁷cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between versions 3.10 and 3.11

Correction of a bug linked to `first-row` and `last-row`.

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -block and, if the creation of the “medium nodes” is required, a node $i-j$ -block-medium is created.

If the user 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 command `\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`.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
@@ commands:	
<code>\@@_Block:</code>	<u>1146</u> , <u>4674</u>
<code>\@@_Block_i</code>	<u>4676</u> , <u>4679</u>
<code>\@@_Block_ii:nnnnn</code>	<u>4679</u> , <u>4680</u>
<code>\@@_Block_iv:nnnnn</code>	<u>4710</u> , <u>4714</u>
<code>\@@_Block_iv:nnnnnn</code>	<u>4872</u> , <u>4874</u>
<code>\@@_Block_v:nnnnn</code>	<u>4711</u> , <u>4809</u>
<code>\@@_Block_v:nnnnnn</code>	<u>4901</u> , <u>4904</u>
<code>\@@_Cdots</code>	<u>1071</u> , <u>1136</u> , <u>3265</u>
<code>\g_@@_Cdots_lines_tl</code>	<u>1163</u> , <u>2486</u>
<code>\@@_Cell:</code>	<u>201</u> , <u>824</u> , <u>1589</u> , <u>1636</u> , <u>1658</u> , <u>2259</u> , <u>3365</u> , <u>3366</u> , <u>3367</u> , <u>3368</u> , <u>3369</u> , <u>3370</u> , <u>3371</u> , <u>3372</u> , <u>3373</u> , <u>3374</u> , <u>3375</u> , <u>3376</u>

<code>\@@_CodeAfter:</code>	1150, 5291	<code>\l_@@_argspec_tl</code>	3247,
<code>\@@_CodeAfter_i:n</code>	826, 2137, 2182, 5291, 5292, 5302		3248, 3249, 3265, 3281, 3297, 3321, 3420,
<code>\@@_CodeAfter_ii:n</code>	5295, 5297		3421, 3422, 3496, 3497, 3498, 3574, 3575, 3576
<code>\@@_CodeAfter_keys:</code>	1228, 2425, 2447	<code>\@@_array:</code>	992, 1936, 1963
<code>\@@_Ddots</code>	1073, 1138, 3297	<code>\c_@@_arydshln_loaded_bool</code>	24, 31, 1575
<code>\g_@@_Ddots_lines_tl</code>	1166, 2484	<code>\l_@@_auto_columns_width_bool</code>	463, 599, 2047, 2051, 4439
<code>\g_@@_HVDotsfor_lines_tl</code>	1168, 2482, 3424, 3500, 5801	<code>\l_@@_baseline_tl</code>	454, 455, 592, 593, 594,
<code>\@@_Hdotsfor:</code>	1076, 1143, 3400		595, 1005, 1397, 1748, 1760, 1765, 1767,
<code>\@@_Hdotsfor:nnnn</code>	3426, 3438		1772, 1777, 1859, 1860, 1864, 1869, 1871, 1876
<code>\@@_Hdotsfor_i</code>	3409, 3415, 3422	<code>\@@_begin_of_NiceMatrix:nn</code>	2248, 2269
<code>\@@_Hline:</code>	1141, 4186	<code>\@@_begin_of_row:</code>	829, 853, 2139
<code>\@@_Hline_i:n</code>	4186, 4187, 4193	<code>\l_@@_block_auto_columns_width_bool</code>	1341, 2052, 4432, 4437, 4447, 4457
<code>\@@_Hline_ii:nn</code>	4190, 4193	<code>\g_@@_block_box_int</code>	306, 1321,
<code>\@@_Hline_iii:n</code>	4191, 4194		4716, 4730, 4732, 4762, 4774, 4786, 4795, 4805
<code>\@@_Hspace:</code>	1142, 3351	<code>\g_@@_blocks_dp_dim</code>	241, 911, 914, 915, 4789, 4792
<code>\@@_Iddots</code>	1074, 1139, 3321	<code>\g_@@_blocks_ht_dim</code>	240, 917, 920, 921, 4780, 4783
<code>\g_@@_Iddots_lines_tl</code>	1167, 2485	<code>\g_@@_blocks_seq</code>	287, 1343, 1798, 4799, 4811, 4872
<code>\@@_Ldots</code>	1070, 1075, 1135, 3249	<code>\g_@@_blocks_wd_dim</code>	239, 905, 908, 909, 4768, 4771
<code>\g_@@_Ldots_lines_tl</code>	1164, 2487	<code>\c_@@_booktabs_loaded_bool</code>	25, 34, 1086, 1830
<code>\l_@@_Matrix_bool</code>	247, 1452, 1468, 2250	<code>\@@_cartesian_path:</code>	3668, 3683, 3798, 3810, 3845
<code>\l_@@_NiceArray_bool</code>	244, 360, 1284, 1353, 1395, 1506, 1518,		3713, 3845, 3846
	2226, 4055, 4056, 4178, 4179, 4385, 4392, 5212	<code>\l_@@_cell_box</code>	831, 877, 879, 885, 891, 894,
<code>\g_@@_NiceMatrixBlock_int</code>	235, 4444, 4449, 4452, 4463		898, 907, 908, 913, 914, 919, 920, 930, 931,
<code>\l_@@_NiceTabular_bool</code>	156,		932, 933, 935, 938, 942, 944, 962, 1088,
	245, 832, 997, 1226, 1230, 1328, 1428,		1239, 1241, 1657, 1668, 2140, 2164, 2167,
	1432, 1507, 1519, 2279, 2288, 4742, 4818, 5220		2169, 2186, 2209, 2213, 4946, 5050, 5084, 5227
<code>\@@_NotEmpty:</code>	1152, 2273	<code>\l_@@_cell_space_bottom_limit_dim</code>	443, 511, 933
<code>\@@_OnlyMainNiceMatrix:n</code>	1148, 3916	<code>\l_@@_cell_space_top_limit_dim</code>	442, 509, 931
<code>\@@_OnlyMainNiceMatrix_i:n</code>	3919, 3926, 3929	<code>\l_@@_cell_type_tl</code>	237, 238, 1589, 1659, 4694, 4696
<code>\@@_SubMatrix</code>	2407, 5424	<code>\@@_cellcolor</code>	1218, 3715, 3727, 3728
<code>\@@_SubMatrix_in_code_before</code>	1225, 5411	<code>\@@_cellcolor_tabular</code>	1080, 3887
<code>\@@_Vdots</code>	1072, 1137, 3281	<code>\g_@@_cells_seq</code>	1974, 1975, 1976, 1978
<code>\g_@@_Vdots_lines_tl</code>	1165, 2483	<code>\@@_chessboardcolors</code>	1224, 3720
<code>\@@_Vdotsfor:</code>	1144, 3498	<code>\@@_cline</code>	135, 1134
<code>\@@_Vdotsfor:nnnn</code>	3502, 3513	<code>\@@_cline_i:nn</code>	136, 137, 149, 152
<code>\@@_W:</code>	1471, 1547	<code>\@@_cline_i:w</code>	137, 138
<code>\@@_actually_color:</code>	1229, 3638	<code>\l_@@_code_before_bool</code>	277, 589, 616, 1012, 1174, 1309, 1349,
<code>\@@_actually_diagbox:nnnnnn</code>	4721, 4938, 5232, 5248		1994, 2011, 2029, 2060, 2086, 2113, 2338, 2441
<code>\@@_actually_draw_Cdots:</code>	2813, 2817	<code>\l_@@_code_before_tl</code>	276, 588, 1228, 1308, 1350
<code>\@@_actually_draw_Ddots:</code>	2963, 2967	<code>\l_@@_code_for_first_col_tl</code>	528, 2151
<code>\@@_actually_draw_Iddots:</code>	3015, 3019	<code>\l_@@_code_for_first_row_tl</code>	532, 841, 5022
<code>\@@_actually_draw_Ldots:</code>	2774, 2778, 3489	<code>\l_@@_code_for_last_col_tl</code>	530, 2195
<code>\@@_actually_draw_Vdots:</code>	2895, 2899, 3564	<code>\l_@@_code_for_last_row_tl</code>	534, 848, 5025
<code>\@@_adapt_S_column:</code>	171, 186, 1327	<code>\l_@@_code_tl</code>	268, 5406, 5603
<code>\@@_add_to_colors_seq:nn</code>	3625, 3637, 3659, 3674, 3689, 3698	<code>\l_@@_col_max_int</code>	295, 2525, 2536, 2604, 2663, 2680
<code>\@@_adjust_pos_of_blocks_seq:</code>	2394, 2449	<code>\l_@@_col_min_int</code>	294, 2530, 2593, 2598, 2661, 2678
<code>\@@_adjust_pos_of_blocks_seq_i:nnnn</code>	2452, 2454	<code>\g_@@_col_total_int</code>	830, 1159, 1381, 2079, 2080, 2116, 2120,
<code>\@@_adjust_size_box:</code>	903, 929, 1667, 2161, 2206		2125, 2126, 2185, 2296, 2299, 2304, 2311,
<code>\@@_adjust_to_submatrix:nn</code>	2658, 2761, 2800, 2881, 2956, 3008		
<code>\@@_adjust_to_submatrix:nnnnnn</code>	2665, 2667		
<code>\@@_after_array:</code>	1461, 2292		
<code>\g_@@_after_col_zero_bool</code>	273, 1038, 2138, 3406		
<code>\@@_analyze_end:Nn</code>	1935, 1980		

2355, 2848, 2866, 2926, 3396, 3397, 3559, 4490, 4500, 4534, 4621, 4884, 5057, 5439, 5477	\g_@@_dp_row_zero_dim
\l_@@_color_tl 876, 877, 1113, 1114, 1407, 1853, 1892
\l_@@_colors_seq 1227, 3628, 3632, 3633, 3642	\@@_draw_Cdots:nnn
\@@_colortbl_like:	2798
\l_@@_colortbl_like_bool 440, 615, 1153, 1494	\@@_draw_Ddots:nnn
\c_@@_colortbl_loaded_bool ... 88, 92, 1103	2954
\l_@@_cols_tl	\@@_draw_Iddots:nnn
... 3667, 3682, 3712, 3737, 3745, 3746, 3848	3006
\g_@@_cols_vlism_seq .. 256, 1489, 1566, 5505	\@@_draw_Ldots:nnn
\@@_columncolor	2759
1223, 3670	\@@_draw_Vdots:nnn
\@@_columncolor:n	2879
3676, 3679	\@@_draw_blocks:
\@@_columncolor_preamble	1798, 4869
1082, 3904	\@@_draw_dotted_lines:
\c_@@_columncolor_regex	2392, 2471
210, 1497	\@@_draw_dotted_lines_i:
\l_@@_columns_width_dim	2474, 2478
.... 236, 600, 735, 2048, 2054, 4445, 4451	\l_@@_draw_first_bool .. 305, 3312, 3336, 3347
\g_@@_com_or_env_str	\@@_draw_hlines:
260, 263	2405, 4175
\@@_computations_for_large_nodes: ...	\@@_draw_line:
.... 4561, 4574, 4579	2796,
\@@_computations_for_medium_nodes: ..	2841, 2952, 3004, 3060, 3062, 3623, 4400, 4430
.... 4481, 4550, 4560, 4571	\@@_draw_line_ii:nn
\@@_compute_a_corner:nnnnnn	3603, 3607
.... 4257, 4259, 4261, 4263, 4268	\@@_draw_line_iii:nn
\@@_compute_corners:	3610, 3614
2393, 4249	\@@_draw_non_standard_dotted_line: ..
\@@_construct_preamble: 3068, 3070
1297, 1465	\@@_draw_non_standard_dotted_line:n ..
\@@_create_col_nodes: 1939, 1967, 1986 3073, 3076
\@@_create_diag_nodes: ... 1208, 2363, 2494	\@@_draw_non_standard_dotted_line:nnn
\@@_create_extra_nodes: 3078, 3083, 3097
1797, 4471	\@@_draw_standard_dotted_line: .. 3067, 3098
\@@_create_large_nodes:	\@@_draw_standard_dotted_line_i: 3161, 3165
4479, 4555	\l_@@_draw_tl
\@@_create_medium_and_large_nodes: ..	299, 4853,
.... 4476, 4566	4857, 4910, 5092, 5098, 5100, 5102, 5139, 5140
\@@_create_medium_nodes:	\@@_draw_vlines:
4477, 4545	2406, 4052
\@@_create_nodes: 4552, 4563, 4573, 4576, 4617	\g_@@_empty_cell_bool 284, 937, 946,
\@@_create_row_node:	2174, 2221, 3263, 3279, 3295, 3319, 3342, 3353
1008, 1041, 1087	\l_@@_empty_corner_cells_seq 2399,
\@@_cut_on_hyphen:w	3990, 3996, 4003, 4115, 4121, 4128, 4251, 4324
.... 3650, 3705, 3710, 3766, 3852, 3853,	\@@_end_Cell:
3869, 3870, 5110, 5119, 5413, 5418, 5431, 5434	203, 924, 1591,
\g_@@_ddots_int	1646, 1663, 2259, 3365, 3366, 3367, 3368,
2373, 2987, 2988	3369, 3370, 3371, 3372, 3373, 3374, 3375, 3376
\@@_def_env:nnn	\l_@@_end_of_row_tl
.... 2232, 2243, 2244, 2245, 2246, 2247 471, 472, 522, 1959, 1960, 5889
\@@_define_L_C_R:	\c_@@_endpgfortikzpicture_tl
223, 1296 43, 47, 2475, 3611, 4371
\c_@@_define_L_C_R_bool ... 222, 1296, 5721	\c_@@_enumitem_loaded_bool
\@@_define_com:nnn 26, 37, 333, 643, 648, 659, 664
.... 5196, 5204, 5205, 5206, 5207, 5208	\@@_env:
\@@_delimiter:nnn	230, 234, 862, 868,
1687, 1695, 5305	963, 969, 1017, 1023, 1029, 1188, 1189,
\l_@@_delimiters_color_tl 474, 713, 715,	1195, 1196, 1203, 1204, 1215, 1995, 1998,
769, 771, 788, 790, 1420, 1421, 1438, 1439,	2000, 2016, 2022, 2025, 2034, 2040, 2043,
5285, 5343, 5344, 5371, 5657, 5658, 5681, 5682	2065, 2071, 2074, 2091, 2097, 2103, 2116,
\l_@@_delimiters_max_width_bool	2120, 2126, 2368, 2416, 2503, 2565, 2633,
.... 475, 574, 712, 1443	2697, 2708, 2721, 2724, 2743, 2746, 2851,
\g_@@_delta_x_one_dim 2375, 2990, 3000	2854, 2869, 2872, 3452, 3470, 3527, 3545,
\g_@@_delta_x_two_dim 2377, 3046, 3056	3596, 3598, 3617, 3620, 4279, 4298, 4316,
\g_@@_delta_y_one_dim 2376, 2992, 3000	4503, 4505, 4513, 4624, 4633, 4651, 4961,
\g_@@_delta_y_two_dim 2378, 3048, 3056	4968, 4972, 4986, 4991, 5003, 5008, 5009,
\@@_diagbox:nn	5012, 5029, 5063, 5320, 5323, 5457, 5459,
1151, 5228	5464, 5466, 5480, 5482, 5487, 5489, 5587, 5598
\@@_dotfill:	\g_@@_env_int .. 229, 230, 232, 1176, 1180,
5217	1183, 1192, 1193, 1200, 1201, 1249, 1252,
\@@_dotfill_i:	1267, 1270, 1340, 1347, 1351, 2303, 2324,
5222, 5224	2342, 2345, 2358, 2437, 3753, 3756, 3781, 4660
\@@_dotfill_ii:	\@@_error:n
5221, 5224, 5225	12,
\@@_dotfill_iii:	336, 361, 484, 494, 544, 669, 718, 728, 734,
5225, 5226	743, 751, 773, 780, 792, 793, 794, 800, 805,
\@@_double_int_eval:n 3570, 3584, 3585	806, 807, 818, 820, 821, 822, 1376, 1386,
\g_@@_dp_ante_last_row_dim	1455, 1782, 1835, 1881, 3740, 4867, 5289,
856, 1119	5391, 5402, 5409, 5440, 5731, 5736, 5763, 5773
\g_@@_dp_last_row_dim	
.... 856, 857, 1122, 1123, 1240, 1241, 1414	

\@@_error:nn	\l_@@_hlines_clist
..... 13, 607, 1555, 1556, 1557, 3252, 310, 536, 550, 581, 1042, 1044,
3255, 3268, 3271, 3284, 3287, 3301, 3302,	1048, 2405, 4181, 4182, 5383, 5443, 5536, 5538
3307, 3308, 3325, 3326, 3331, 3332, 4265, 5396	\g_@@_ht_last_row_dim
\@@_error:nnn 858, 1120, 1121, 1238, 1239, 1413
..... 14, 3601, 5534, 5569	\g_@@_ht_row_one_dim ..
\@@_error_too_much_cols:	884, 885, 1117, 1118
..... 1528, 5786	\g_@@_ht_row_zero_dim
\@@_everycr: 878, 879, 1115, 1116, 1408, 1852, 1891
..... 1034, 1108, 1111	\@@_i:
\@@_everycr_i:	4483, 4485,
..... 1034, 1035	4486, 4487, 4488, 4497, 4503, 4505, 4506,
\l_@@_except_corners_clist	4507, 4508, 4513, 4514, 4515, 4516, 4524,
..... 459, 575, 579, 3956, 4082, 4252	4527, 4529, 4530, 4531, 4583, 4585, 4588,
\l_@@_exterior_arraycolsep_bool	4589, 4593, 4594, 4619, 4624, 4626, 4628,
..... 456, 731, 1509, 1521	4632, 4633, 4644, 4651, 4653, 4655, 4659, 4660
\l_@@_extra_left_margin_dim	\g_@@_iddots_int
..... 469, 566, 1300, 2172	2374, 3043, 3044
\l_@@_extra_right_margin_dim	\l_@@_in_env_bool
..... 470, 567, 1368, 2217, 2929	243, 360, 1331, 1332
\@@_extract_coords_values:	\c_@@_in_preamble_bool .
4642, 4649	21, 22, 23, 639, 655
\@@_fatal:n	\l_@@_initial_i_int
15, 252, 1331, 1683, 1693,	2380, 2510,
1944, 1948, 1950, 1983, 3411, 5791, 5794, 5797	2585, 2588, 2613, 2621, 2625, 2634, 2642,
\@@_fatal:nn	2652, 2698, 2783, 2829, 2831, 2845, 2851,
..... 16, 1580	2854, 3023, 3442, 3443, 3453, 3521, 3531, 3533
\l_@@_fill_tl	\l_@@_initial_j_int
298, 4851, 4923, 4928 2381, 2511, 2586, 2593, 2598,
\l_@@_final_i_int	2604, 2614, 2622, 2626, 2635, 2643, 2653,
..... 2382, 2512, 2517, 2520, 2545, 2553,	2699, 2721, 2724, 2732, 2919, 2921, 2926,
2557, 2566, 2574, 2654, 2709, 2790, 2863,	2972, 3026, 3446, 3456, 3458, 3517, 3518, 3529
2869, 2872, 3033, 3443, 3471, 3539, 3549, 3551	\l_@@_initial_open_bool
\l_@@_final_j_int 2384, 2587, 2591, 2594, 2601, 2607,
2383,	2611, 2627, 2780, 2819, 2826, 2836, 2902,
2513, 2518, 2525, 2530, 2536, 2546, 2554,	2909, 2915, 2969, 3021, 3167, 3214, 3440,
2558, 2567, 2575, 2655, 2710, 2743, 2746,	3447, 3459, 3515, 3522, 3534, 3592, 4367, 4404
2754, 2980, 3036, 3464, 3474, 3476, 3518, 3547	\@@_insert_tabularnotes:
\l_@@_final_open_bool	1802, 1805
2385, 2519,	\@@_instruction_of_type:nnn
2523, 2526, 2533, 2539, 2543, 2559, 2787, 973, 3257, 3273, 3289, 3312, 3336
2822, 2827, 2838, 2902, 2912, 2917, 2938,	\c_@@_integers_alist_tl
2977, 3031, 3169, 3184, 3215, 3216, 3441,	5615, 5626
3465, 3477, 3516, 3540, 3552, 3593, 4368, 4405	\l_@@_inter_dots_dim
\@@_find_extremities_of_line:nnnn 444, 445, 2389, 3172, 3179, 3190, 3198,
..... 2507, 2764, 2803, 2884, 2959, 3011	3205, 3210, 3222, 3230, 4396, 4398, 4426, 4428
\l_@@_first_col_int	\g_@@_internal_code_after_tl
123, 136, 314, 269, 1625, 1686, 1694, 1714,
315, 524, 798, 829, 1390, 1501, 1989, 2009,	2421, 2422, 4201, 4360, 4719, 4936, 5230, 5421
2349, 2848, 2866, 3404, 3860, 3918, 4490,	\@@_intersect_our_row:nnnn
4500, 4534, 4582, 4621, 5178, 5184, 5190, 5477	3834
\l_@@_first_i_tl	\@@_intersect_our_row_p:nnnn
5427, 5432, 5453, 5471,	3785
5480, 5482, 5537, 5544, 5546, 5630, 5641, 5645	\@@_j:
\l_@@_first_j_tl	4490, 4492,
5428, 5433, 5457,	4493, 4494, 4495, 4500, 4503, 4505, 4508,
5459, 5507, 5520, 5527, 5529, 5631, 5642, 5646	4510, 4511, 4513, 4516, 4518, 4519, 4534,
\l_@@_first_row_int	4537, 4539, 4540, 4541, 4596, 4598, 4601,
312, 313,	4603, 4607, 4608, 4621, 4624, 4625, 4627,
525, 802, 1157, 1405, 1779, 1850, 1878,	4632, 4633, 4645, 4651, 4652, 4654, 4659, 4660
1889, 2347, 2718, 2740, 4483, 4497, 4524,	\l_@@_l_dim
4581, 4619, 4965, 4983, 5176, 5317, 5454, 5907 3145, 3146, 3159, 3160, 3172, 3178,
\c_@@_footnote_bool	3189, 3197, 3205, 3210, 3222, 3223, 3230, 3231
1316, 1463, 5698, 5734, 5757, 5760, 5770, 5776	\l_@@_large_nodes_bool ..
\c_@@_footnotehyper_bool .	466, 557, 4475, 4479
5697, 5735, 5767	\g_@@_last_col_found_bool ..
\@@_full_name_env:	322, 1162,
..... 261, 5807, 5814, 5822, 5830, 5834,	1382, 1447, 2078, 2107, 2183, 2295, 2352, 5055
5921, 5925, 5928, 5941, 5946, 5951, 5953,	\l_@@_last_col_int
5970, 5989, 5994, 5999, 6006, 6025, 6034, 6204 320, 321, 719, 762, 764, 781, 801,
\@@_hdottedline:	819, 1186, 1263, 1269, 1276, 1385, 1513,
1140, 4353	2255, 2257, 2296, 2299, 2351, 2889, 2924,
\@@_hdottedline:n	3254, 3270, 3308, 3332, 4877, 4882, 4883,
4361, 4365	4884, 4887, 4918, 4930, 4942, 4957, 4986,
\@@_hdottedline_i:	4991, 4999, 5014, 5180, 5186, 5192, 5790, 5808
4356, 4358	
\@@_hdottedline_i:n	
4370, 4374	
\@@_hline:nn	
4063, 4183, 4202	
\@@_hline_i:nn	
2403, 4066, 4069	
\@@_hline_i_complete:nn	
2403, 4173	
\@@_hline_ii:nnnn ...	
4091, 4102, 4135, 4174	

<code>\l_@@_last_col_without_value_bool</code> ...	<code>\l_@@_name_str</code>
..... 319, 761, 2297, 5793 464, 609, 864, 867, 965, 968, 1025,
<code>\l_@@_last_empty_column_int</code>	1028, 1247, 1256, 1259, 1265, 1274, 1277,
..... 4289, 4290, 4303, 4309, 4322	1999, 2000, 2024, 2025, 2042, 2043, 2073,
<code>\l_@@_last_empty_row_int</code>	2074, 2099, 2102, 2122, 2125, 2306, 2310,
..... 4271, 4272, 4285, 4306	2327, 2331, 2364, 2368, 4629, 4632, 4656, 4659
<code>\l_@@_last_i_tl</code>	<code>\g_@@_names_seq</code>
5429,	242, 606, 608, 6313
5435, 5438, 5453, 5474, 5487, 5489, 5537, 5544	<code>\l_@@_nb_cols_int</code>
<code>\l_@@_last_j_tl</code> 5167, 5172, 5175, 5179, 5185, 5191
5430, 5436, 5439, 5464, 5466, 5510, 5520, 5527	<code>\l_@@_nb_rows_int</code>
<code>\l_@@_last_row_int</code>	5166, 5171, 5182
..... 316, 317, 526, 846, 892, 1054,	<code>\@@_newcolumnntype</code>
1184, 1234, 1244, 1251, 1258, 1370, 1374,	1061, 1470, 1471
1377, 1389, 1411, 1961, 1962, 2147, 2148,	<code>\@@_node_for_multicolumn:nn</code>
2192, 2193, 2318, 2769, 2808, 3286, 3302,	4640, 4647
3326, 3924, 3932, 4876, 4879, 4880, 4899,	<code>\@@_node_for_the_cell:</code> 943, 949, 2168, 2218
4918, 4930, 4941, 4955, 5013, 5024, 5188, 6024	<code>\@@_node_left:nn</code>
<code>\l_@@_last_row_without_value_bool</code> ...	5586, 5587, 5649
..... 318, 1246, 1372, 2316	<code>\@@_node_position:</code> .. 1195, 1197, 1203, 1205
<code>\l_@@_left_delim_dim</code>	<code>\@@_node_right:nn</code>
..... 1282, 1286, 1291, 1927, 2170	5596, 5598, 5673
<code>\l_@@_left_delim_tl</code> .. 1290, 1318, 1422, 4395	<code>\g_@@_not_empty_cell_bool</code> 275, 941, 947, 2274
<code>\l_@@_left_margin_dim</code>	<code>\@@_not_in_exterior:nnnn</code>
..... 467, 560, 1299, 2171, 4386, 4612	3826
<code>\l_@@_letter_for_dotted_lines_str</code> ...	<code>\@@_not_in_exterior_p:nnnn</code>
..... 742, 753, 754, 1561	3758
<code>\l_@@_letter_vlism_tl</code>	<code>\l_@@_notes_above_space_dim</code>
255, 543, 1564	460, 461
<code>\l_@@_light_syntax_bool</code>	<code>\l_@@_notes_bottomrule_bool</code>
..... 453, 520, 1302, 1363, 2319 627, 784, 813, 1828
<code>\@@_light_syntax_i</code>	<code>\l_@@_notes_code_after_tl</code>
1952, 1955	625, 1837
<code>\@@_line</code>	<code>\l_@@_notes_code_before_tl</code>
2420, 3576	623, 1809
<code>\@@_line_i:nn</code>	<code>\@@_notes_label_in_list:n</code> 329, 348, 356, 635
3583, 3590	<code>\@@_notes_label_in_tabular:n</code> . 328, 369, 632
<code>\l_@@_line_width_dim</code>	<code>\l_@@_notes_para_bool</code> .. 621, 782, 811, 1813
..... 302, 4859, 4911, 5093, 5131, 5142	<code>\@@_notes_style:n</code>
<code>\@@_line_with_light_syntax:n</code> ... 1966, 1970 327, 330, 348, 356, 372, 377, 629
<code>\@@_line_with_light_syntax_i:n</code>	<code>\l_@@_nullify_dots_bool</code>
1965, 1971, 1972 462, 555, 3261, 3277, 3293, 3317, 3340
<code>\@@_math_toggle_token:</code>	<code>\l_@@_number_of_notes_int</code> 326, 363, 373, 383
155, 926, 2141, 2158, 2187, 2203, 5272, 5276	<code>\@@_old_CT@arc@</code>
<code>\g_@@_max_cell_width_dim</code>	1333, 2445
..... 934, 935, 1342, 2053, 4438, 4464	<code>\@@_old_cdots</code>
<code>\c_@@_max_l_dim</code>	1128, 3278
3159, 3164	<code>\@@_old_ddots</code>
<code>\l_@@_medium_nodes_bool</code> 465, 556, 4473, 5005	1130, 3318
<code>\@@_message_hdotsfor:</code> 5799, 5807, 5814, 5822	<code>\@@_old_dotfill</code>
<code>\@@_msg_new:nn</code>	5216, 5219, 5227
. 17, 5713, 5739, 5748, 5804, 5811, 5819,	<code>\@@_old_dotfill:</code>
5827, 5832, 5838, 5843, 5848, 5853, 5858,	1149
5863, 5869, 5875, 5880, 5886, 5892, 5897,	<code>\l_@@_old_iRow_int</code>
5904, 5911, 5918, 5924, 5926, 5932, 5949,	270, 1090, 2491
5956, 5961, 5968, 5975, 5982, 5987, 5997,	<code>\@@_old_ialign:</code>
6003, 6010, 6016, 6022, 6030, 6032, 6038, 6315	1007, 1124, 4871
<code>\@@_msg_new:nnn</code> ... 18, 5699, 5938, 6043,	<code>\@@_old_iddots</code>
6053, 6068, 6089, 6107, 6150, 6201, 6252, 6301	1131, 3341
<code>\@@_msg_redirect_name:nn</code>	<code>\l_@@_old_jCol_int</code>
..... 19, 737, 1459, 4890, 4893	271, 1093, 2492
<code>\@@_multicolumn:nnn</code>	<code>\@@_old_ldots</code>
1145, 3357	1127, 3262
<code>\g_@@_multicolumn_cells_seq</code>	<code>\@@_old_multicolumn</code>
... 1155, 3384, 4508, 4516, 4638, 4970, 4988	3356, 3360
<code>\g_@@_multicolumn_sizes_seq</code> 1156, 3386, 4639	<code>\@@_old_pgfpaintanchor</code>
<code>\g_@@_name_env_str</code>	163, 5607, 5611
259,	<code>\@@_old_pgfulil@check@rerun</code>
264, 265, 1325, 1326, 1982, 2227, 2228,	81, 85
2236, 2237, 2266, 2277, 2285, 2443, 5200, 5788	<code>\@@_old_vdots</code>
	1129, 3294
	<code>\@@_open_x_final_dim:</code>
 2737, 2789, 2823, 2981, 3038
	<code>\@@_open_x_initial_dim:</code>
 2715, 2782, 2820, 2974, 3028
	<code>\@@_open_y_final_dim:</code> 2861, 2913, 2979, 3035
	<code>\@@_open_y_initial_dim:</code>
 2843, 2910, 2971, 3025
	<code>\l_@@_parallelize_diags_bool</code>
 457, 458, 552, 2371, 2985, 3041
	<code>\@@_patch_preamble:n</code>
	1491, 1532,
	1570, 1578, 1599, 1628, 1688, 1708, 1716, 1736
	<code>\@@_patch_preamble_i:n</code> 1536, 1537, 1538, 1585
	<code>\@@_patch_preamble_ii:nn</code>
 1539, 1540, 1541, 1596
	<code>\@@_patch_preamble_iii:n</code> . 1542, 1601, 1609
	<code>\@@_patch_preamble_iii_i:n</code> 1604, 1606

\@@_patch_preamble_iv:nnn	\l_@@_real_right_delim_dim 1899, 1925, 1931
..... 1543, 1544, 1545, 1631	\@@_rectanglecolor 1219, 3685, 3718
\@@_patch_preamble_ix:n 1721, 1739	\@@_rectanglecolor:nnn ... 3691, 3700, 3703
\@@_patch_preamble_v:nnnn 1546, 1547, 1652	\@@_renew_NC@rewrite@S: 192, 194, 1161
\@@_patch_preamble_vi:n 1548, 1674	\@@_renew_dots: 1068, 1154
\@@_patch_preamble_vii:n	\l_@@_renew_dots_bool
..... 1549, 1550, 1551, 1680 553, 727, 1154, 5723, 5730
\@@_patch_preamble_viii:n	\@@_renew_matrix: 722, 726, 5146, 5725, 5729
..... 1552, 1553, 1554, 1690	\l_@@_respect_blocks_bool 3735, 3750, 3778
\@@_patch_preamble_viii_i:n 1696, 1698	\@@_restore_iRow_jCol: 2444, 2489
\@@_patch_preamble_x:n	\@@_revtex_array: 984, 995
..... 1594, 1650, 1672, 1678, 1718, 1742	\c_@@_revtex_bool 50, 52, 55, 57, 994
\@@_patch_preamble_xi:n 1562, 1710	\l_@@_right_delim_dim
\@@_pgf_rect_node:nnn 416, 5007 1283, 1287, 1293, 1930, 2215
\@@_pgf_rect_node:nnnnn	\l_@@_right_delim_tl . 1292, 1319, 1440, 4397
..... 391, 4623, 4650, 4960, 5002, 5573	\l_@@_right_margin_dim
\c_@@_pgfortikzpicture_tl 468, 562, 1367, 2216, 2928, 4393, 4615
..... 42, 46, 2473, 3609, 4369	\@@_rotate: 1147, 3569
\@@_pgfpointanchor:n 5601, 5608	\g_@@_rotate_bool
\@@_pgfpointanchor_i:nn 5611, 5613 248, 901, 928, 1666, 2160,
\@@_pgfpointanchor_ii:w 5614, 5622	2205, 3569, 4741, 4759, 4764, 4824, 4837, 4947
\@@_pgfpointanchor_iii:w 5635, 5637	\@@_rotate_cell_box:
\@@_picture_position: 1189, 1197, 1205 889, 928, 1666, 2160, 2205, 4947
\l_@@_pos_of_block_tl ... 303, 304, 4665,	\l_@@_rounded_corners_dim
4667, 4669, 4695, 4696, 4698, 4741, 4745, 300, 4855, 4931, 5107, 5108, 5143
4752, 4802, 4825, 4827, 4838, 4841, 4861,	\@@_roundedrectanglecolor 1220, 3694
4863, 4865, 5031, 5043, 5054, 5058, 5065, 5077	\l_@@_row_max_int 293, 2520, 2662, 2679
\g_@@_pos_of_blocks_seq 288, 1344, 2359,	\l_@@_row_min_int 292, 2588, 2660, 2677
2397, 2451, 3387, 3950, 4076, 4336, 4906, 5240	\g_@@_row_of_col_done_bool
\g_@@_pos_of_stroken_blocks_seq 274, 1039, 1324, 2008
..... 290, 1345, 3954, 4080, 4920	\g_@@_row_total_int
\g_@@_pos_of_xdots_seq	1158, 1388, 1780, 1879, 2318, 2325, 2332,
..... 289, 1346, 2398, 2650, 3952, 4078	2348, 2718, 2740, 3484, 4483, 4497, 4524,
\@@_pre_array: 1171, 1310, 1360	4619, 4899, 4965, 4983, 5317, 5438, 5454, 5908
\@@_pre_array_i:w 1306, 1360	\@@_rowcolor 1221, 3655
\@@_pre_array_ii: 1084, 1281	\@@_rowcolor:n 3661, 3664
\c_@@_preamble_first_col_tl 1502, 2133	\@@_rowcolor_tabular 1081, 3895
\c_@@_preamble_last_col_tl 1514, 2178	\@@_rowcolors 1222, 3742
\g_@@_preamble_tl	\@@_rowcolors_i:nnnn 3786, 3821
..... 1320, 1472, 1476, 1479, 1485,	\l_@@_rowcolors_restart_bool ... 3738, 3769
1499, 1502, 1511, 1514, 1523, 1527, 1568,	\g_@@_rows_seq . 1958, 1960, 1962, 1964, 1966
1577, 1587, 1598, 1611, 1633, 1654, 1676,	\l_@@_rows_tl .. 3666, 3681, 3711, 3788, 3865
1685, 1707, 1712, 1725, 1732, 1741, 1936, 1963	\l_@@_rules_color_tl
\@@_pred:n 272, 498, 1357, 1358, 5502, 5503
.... 124, 154, 2257, 3991, 4004, 4116, 4129	\@@_set_CT@arc@: 157, 1358, 5503
\@@_provide_pgfsyspdfmark: . 211, 220, 1315	\@@_set_CT@arc@_i: 158, 159
\@@_put_box_in_flow: 1445, 1744, 1929	\@@_set_CT@arc@_ii: 158, 161
\@@_put_box_in_flow_bis:nn 1444, 1896	\@@_set_final_coords: 2688, 2713
\@@_put_box_in_flow_i: 1750, 1752	\@@_set_final_coords_from_anchor:n ..
\@@_qpoint:n 2704, 2793, 2824, 2905, 2914, 2984, 3040
..... 233, 1755, 1757, 1769, 1785, 1844,	\@@_set_initial_coords: 2683, 2702
1846, 1862, 1873, 1884, 2500, 2502, 2732,	\@@_set_initial_coords_from_anchor:n .
2754, 2783, 2790, 2829, 2831, 2845, 2863,	... 2693, 2786, 2821, 2904, 2911, 2976, 3030
2919, 2921, 2972, 2980, 3023, 3026, 3033,	\@@_set_size:n 5164, 5173
3036, 3617, 3620, 3859, 3863, 3876, 3878,	\c_@@_siunitx_loaded_bool 164, 168, 173, 191
4014, 4016, 4018, 4139, 4141, 4143, 4377,	\l_@@_small_bool 720, 767, 777,
4381, 4388, 4422, 4425, 4427, 4529, 4539,	803, 835, 1096, 1682, 1692, 2142, 2188, 2386
4951, 4953, 4955, 4957, 4979, 4999, 5027,	\@@_standard_cline 120, 1133
5115, 5117, 5124, 5126, 5253, 5255, 5258,	\@@_standard_cline:w 120, 121
5260, 5310, 5312, 5471, 5474, 5512, 5529, 5546	\l_@@_standard_cline_bool .. 441, 507, 1132
\l_@@_radius_dim 448, 449, 1713,	\c_@@_standard_tl 451, 452, 3066, 4399, 4429
2388, 2794, 2795, 3239, 4355, 4379, 4423, 4424	\g_@@_static_num_of_col_int
\l_@@_real_left_delim_dim 1898, 1913, 1928 297, 1454, 1492, 4887, 5823, 5835, 5990

<code>\l_@@_stop_loop_bool</code>	2514, 2515, 2547, 2560, 2569, 2582, 2583, 2615, 2628, 2637
<code>@@_stroke_block:nnn</code>	4915, 5089
<code>\l_@@_submatrix_extra_height_dim</code>	307, 5363, 5497
<code>\l_@@_submatrix_left_xshift_dim</code>	308, 5365, 5549, 5582
<code>\l_@@_submatrix_name_str</code>	5398, 5442, 5571, 5573, 5585, 5587, 5595, 5598
<code>\g_@@_submatrix_names_seq</code>	291, 2424, 5395, 5399, 5947
<code>\l_@@_submatrix_right_xshift_dim</code>	309, 5367, 5558, 5592
<code>\g_@@_submatrix_seq</code>	296, 1173, 2664, 5419
<code>\l_@@_submatrix_slim_bool</code>	5373, 5452
<code>@@_succ:n</code>	149, 153, 1017, 1023, 1049, 1626, 1687, 1730, 1757, 2091, 2097, 2102, 2103, 2116, 2120, 2125, 2126, 2353, 2754, 2831, 2921, 2980, 3026, 3033, 3830, 3863, 3876, 3987, 4018, 4056, 4112, 4143, 4179, 4202, 4341, 4343, 4345, 4347, 4388, 4427, 4589, 4593, 4603, 4607, 4955, 4957, 4999, 5124, 5126, 5258, 5260, 5312
<code>\l_@@_suffix_tl</code>	4551, 4562, 4572, 4575, 4624, 4632, 4633, 4651, 4659, 4660
<code>\c_@@_table_collect_begin_tl</code>	181, 183, 201
<code>\c_@@_table_print_tl</code>	184, 185, 203
<code>\l_@@_tabular_width_dim</code>	246, 1000, 1002, 1525, 2286
<code>\l_@@_tabularnote_tl</code>	325, 786, 815, 1801, 1810
<code>\g_@@_tabularnotes_seq</code>	324, 364, 1816, 1822, 1838
<code>@@_test_hline_in_block:nnnn</code>	4077, 4079, 4205
<code>@@_test_hline_in_stroken_block:nnnn</code>	4081, 4227
<code>@@_test_if_cell_in_a_block:nn</code>	4275, 4293, 4311, 4331
<code>@@_test_if_cell_in_block:nnnnnnn</code>	4337, 4339
<code>@@_test_if_math_mode:</code>	249, 1330, 2238
<code>@@_test_in_corner_h:</code>	4082, 4110
<code>@@_test_in_corner_v:</code>	3957, 3985
<code>@@_test_vline_in_block:nnnn</code>	3951, 3953, 4216
<code>@@_test_vline_in_stroken_block:nnnn</code>	3955, 4238
<code>\l_@@_the_array_box</code>	1295, 1298, 1795, 1796
<code>\c_@@_tikz_loaded_bool</code>	27, 41, 1210, 2408, 4406
<code>@@_true_c:</code>	202, 1548
<code>\l_@@_type_of_col_tl</code>	765, 766, 2267, 2269
<code>\c_@@_types_of_matrix_seq</code>	5778, 5779, 5784, 5788
<code>@@_update_for_first_and_last_row:</code>	872, 936, 1236, 2162, 2207
<code>@@_use_arraybox_with_notes:</code>	1402, 1857
<code>@@_use_arraybox_with_notes_b:</code>	1399, 1841
<code>@@_use_arraybox_with_notes_c:</code>	1400, 1431, 1793, 1855, 1894
<code>@@_vdottedline:n</code>	1715, 4402
<code>@@_vdottedline_i:n</code>	4409, 4414, 4418
<code>@@_vline:nn</code>	1626, 3934, 4060
<code>@@_vline_i:nn</code>	2402, 3939, 3943
<code>@@_vline_i_complete:nn</code>	2402, 4050
<code>@@_vline_ii:nnnn</code>	3966, 3977, 4010, 4051
<code>\l_@@_vlines_clist</code>	311, 537, 549, 580, 1477, 1483, 1508, 1520, 1723, 1730, 2406, 4058, 4059, 5385, 5444, 5519, 5521
<code>@@_w:</code>	1470, 1546
<code>\g_@@_width_first_col_dim</code>	286, 1323, 1393, 2003, 2163, 2164
<code>\g_@@_width_last_col_dim</code>	285, 1322, 1449, 2112, 2208, 2209
<code>\l_@@_x_final_dim</code>	280, 2690, 2739, 2748, 2749, 2752, 2755, 2756, 2907, 2923, 2931, 2935, 2939, 2941, 2946, 2948, 2982, 2991, 2999, 3037, 3047, 3055, 3094, 3109, 3118, 3152, 3204, 3220, 3621, 4389, 4398, 4424, 5451, 5467, 5468, 5558, 5575, 5592
<code>\l_@@_x_initial_dim</code>	278, 2685, 2717, 2726, 2727, 2730, 2733, 2734, 2907, 2922, 2923, 2930, 2935, 2939, 2941, 2943, 2946, 2948, 2973, 2991, 2999, 3027, 3047, 3055, 3091, 3108, 3118, 3152, 3204, 3218, 3220, 3238, 3240, 3618, 4382, 4396, 4423, 5450, 5460, 5461, 5549, 5574, 5582
<code>\l_@@_xdots_color_tl</code>	473, 487, 2773, 2812, 2893, 2894, 2962, 3014, 3074, 3488, 3563, 3580
<code>\l_@@_xdots_down_tl</code>	491, 3081, 3102, 3137
<code>\l_@@_xdots_line_style_tl</code>	450, 452, 483, 3066, 3074, 4399, 4429
<code>\l_@@_xdots_shorten_dim</code>	446, 447, 489, 2390, 3088, 3089, 3178, 3189, 3197
<code>\l_@@_xdots_up_tl</code>	492, 3080, 3101, 3127
<code>\l_@@_y_final_dim</code>	281, 2691, 2791, 2795, 2833, 2837, 2839, 2864, 2874, 2875, 2993, 2996, 3034, 3049, 3052, 3094, 3109, 3117, 3154, 3209, 3228, 3622, 4380, 4428, 5313, 5335, 5350, 5475, 5490, 5491, 5496, 5514, 5531, 5575, 5583, 5593
<code>\l_@@_y_initial_dim</code>	279, 2686, 2784, 2794, 2832, 2833, 2837, 2839, 2846, 2856, 2857, 2993, 2998, 3024, 3049, 3054, 3091, 3108, 3117, 3154, 3209, 3226, 3228, 3238, 3241, 3619, 4378, 4379, 4380, 4426, 5311, 5335, 5350, 5472, 5483, 5484, 5496, 5513, 5530, 5574, 5583, 5593
<code>\</code>	1949, 1971, 5180, 5186, 5192, 5701, 5702, 5745, 5754, 5835, 5840, 5845, 5850, 5855, 5883, 5888, 5894, 5901, 5908, 5914, 5915, 5929, 5935, 5941, 5942, 5953, 5965, 5971, 5978, 5985, 5994, 6000, 6007, 6013, 6019, 6031, 6035, 6040, 6046, 6055, 6056, 6070, 6071, 6087, 6091, 6092, 6110, 6111, 6153, 6154, 6204, 6205, 6255, 6256, 6308, 6309
<code>\{</code>	265, 1551, 1704, 2245, 5208, 5554, 5934, 6007, 6153, 6255
<code>\}</code>	265, 1554, 2245, 5208, 5563, 5934, 6007, 6153, 6255
<code>\ </code>	2247, 5207
<code>\sq</code>	5802, 5807, 5814, 5822, 5823, 5834, 5835, 5882, 5907, 5908, 5921, 5925, 5928, 5940, 5946, 5958, 5989, 5990, 5991, 5999, 6005, 6018, 6025, 6026, 6034

A	
\A	5393
\aboverulesep	1832
\addtocounter	381
\alph	327
\arraybackslash	1639
\arraycolsep	561, 563, 565, 999, 1099, 1286, 1287, 1430, 1434, 4385, 4392
\arrayrulecolor	95
\arrayrulewidth	128, 133, 145, 500, 863, 1016, 1018, 1024, 1055, 1480, 1486, 1569, 1619, 1726, 1733, 1849, 1888, 2015, 2017, 2023, 2033, 2035, 2041, 2064, 2066, 2072, 2090, 2092, 2098, 3861, 3862, 3864, 3877, 3879, 4026, 4027, 4029, 4040, 4046, 4152, 4163, 4169, 4198, 4464, 5093, 5335, 5497, 5501
\arraystretch	1098, 2847, 2865, 4738, 4821, 4834, 5473, 5476
\AtBeginDocument	23, 28, 73, 89, 165, 189, 331, 445, 447, 449, 461, 641, 657, 2469, 3245, 3418, 3494, 3572, 3605, 4363
\AutoNiceMatrix	5209
\AutoNiceMatrixWithDelims	5169, 5201, 5213
B	
\baselineskip	98, 105
\bgroup	1317
\bigskip	1359
\Block	1146, 6046
\BNiceMatrix	5161
\bNiceMatrix	5158
\Body	1306
bool commands:	
\bool_do_until:Nn	2515, 2583
\bool_gset_false:N	901, 946, 947, 1038, 1162, 1324, 2174, 2221, 4214, 4225, 4236, 4247, 4764
\bool_gset_true:N	2008, 2138, 2183, 2274, 3263, 3279, 3295, 3319, 3342, 3353, 3569, 3949, 4075
\bool_if:NnTF	156, 173, 643, 648, 659, 664, 832, 835, 928, 1012, 1039, 1086, 1096, 1153, 1154, 1174, 1210, 1226, 1230, 1296, 1316, 1331, 1341, 1372, 1447, 1452, 1463, 1468, 1494, 1666, 1682, 1692, 1828, 1994, 2011, 2029, 2047, 2060, 2086, 2107, 2113, 2142, 2160, 2188, 2205, 2295, 2297, 2316, 2319, 2338, 2371, 2386, 2408, 2836, 2838, 2985, 3041, 3261, 3277, 3293, 3317, 3340, 4284, 4302, 4320, 4447, 4457, 4479, 4741, 4759, 4824, 4837, 4947, 5005, 5220, 5757, 5767, 5793
\bool_if:nTF	191, 333, 360, 975, 1382, 2669, 3594, 3836, 4473, 5055, 5314, 5324, 5326, 5339, 5345, 5354
\bool_lazy_all:nTF	1504, 1516, 2395, 4207, 4218, 4229, 4240
\bool_lazy_and:nnTF	1573, 2050, 2143, 2350, 2825, 3100, 3402, 3749, 3777, 4020, 4145, 5111, 5523, 5540
\bool_lazy_any:nTF	1700
\bool_lazy_or:nnTF	480, 940, 1778, 1799, 1877, 2191, 2902, 3158, 3828, 4276, 4294, 4312, 4682, 4687, 4707, 4909, 5437

\bool_lazy_or_p:nn	2146
\bool_not_p:n	1507, 1509, 1519, 1521, 2052, 2352
\bool_set:Nn	2906, 3771
\c_false_bool	1695, 3257, 3273, 3289
\g_tmpa_bool	3949, 3958, 3992, 4000, 4005, 4075, 4083, 4117, 4125, 4130, 4214, 4225, 4236, 4247
\l_tmpb_bool	4281, 4295, 4313, 4335, 4348
\c_true_bool	1687
box commands:	
\box_clear_new:N	1088, 1295
\box_dp:N	857, 877, 914, 933, 1114, 1123, 1241, 1644, 1747, 1907, 1920, 2865, 4794, 5476
\box_gc_clear_new:N	4729
\box_grotate:Nn	4761
\box_ht:N	858, 879, 885, 897, 920, 931, 1116, 1118, 1121, 1239, 1640, 1746, 1907, 1920, 2847, 4785, 5473
\box_move_up:nn	64, 66, 68, 1790, 1855, 1894
\box_rotate:Nn	891
\box_set_dp:Nn	913, 932, 1747
\box_set_ht:Nn	919, 930, 1746
\box_set_wd:Nn	907
\box_use:N	384, 898
\box_use_drop:N	938, 944, 962, 1668, 1749, 1790, 1791, 1796, 2169, 4804, 5050, 5084
\box_wd:N	385, 908, 935, 942, 1291, 1293, 1795, 1914, 1926, 2164, 2167, 2209, 2213, 4773, 5227
\l_tmpa_box	367, 384, 385, 1290, 1291, 1292, 1293, 1417, 1746, 1747, 1749, 1790, 1791, 1907, 1920
\l_tmpb_box	1900, 1914, 1915, 1926
C	
\c	210, 1498
\Cdots	1136, 3268, 3271
\cdots	1071, 1128
\cellcolor	1080, 1218, 3891
\chessboardcolors	1224
\cline	148, 1133, 1134
clist commands:	
\clist_clear:N	5443, 5444
\clist_if_empty:NnTF	3956, 4082
\clist_if_in:NnTF	1047, 1483, 1730, 4059, 4182
\clist_map_inline:Nn	3848, 3865, 4252, 5521, 5538
\clist_map_inline:nn	2262, 3717, 3762
\clist_new:N	310, 311, 459
\clist_set:Nn	549, 550, 579, 580, 581
\CodeAfter	826, 1150, 1952, 1955, 2137, 2182, 2423, 6058
\CodeBefore	1313
\color	99, 106, 160, 162, 1421, 1439, 2767, 2770, 2773, 2806, 2809, 2812, 2887, 2890, 2894, 2962, 3014, 3482, 3485, 3488, 3557, 3560, 3563, 3580, 3644, 3795, 3796, 3807, 3808, 4736, 4857, 5344, 5658, 5682
\colorlet	257, 258, 842, 849, 2152, 2196
\columncolor	1082, 1223, 2432, 3910
\cr	132, 150, 2131
\crrcr	1988

cs commands:		\draw 3085
\cs_generate_variant:Nn 58, 152, 3097, 3637, 4469, 4470		E
\cs_gset:Npn 99, 106, 2303, 2310, 2324, 2331, 4462		\egroup 1462
\cs_gset_eq:NN ... 186, 220, 1107, 1333, 2445		else commands:
\cs_if_exist:NTF 57, 1089, 1092, 1249, 1256, 1267, 1274, 1334, 1337, 2491, 2492, 2550, 2563, 2618, 2631, 2720, 2742, 2850, 2868, 3450, 3468, 3525, 3543, 4449, 4502, 4967, 4985, 5319, 5456, 5463, 5479, 5486		\else: 251
\cs_if_exist_p:N 481, 3752, 3780, 4278, 4297, 4315		\endarray 1940, 1968
\cs_if_free:NTF 216, 2762, 2801, 2882, 2957, 3009		\endBNiceMatrix 5162
\cs_if_free_p:N 3596, 3598		\endbNiceMatrix 5159
\cs_new_protected:Npx 2471, 3607, 4365		\endNiceArray 2282, 2291
\cs_set:Nn 629, 632, 635		\endNiceArrayWithDelims 2231, 2241
\cs_set:Npn 95, 96, 102, 103, 108, 120, 121, 135, 137, 160, 162, 330, 1063, 2509, 2571, 2639, 3492, 3567, 4186, 4187, 4193, 4194, 4738, 4821, 4834		\endpgfscope 3142, 5273
\cs_set_nopar:Npn 989, 1001, 1098, 1101, 4644, 4645		\endpNiceMatrix 5150
\cs_set_nopar:Npx 1002		\endsavenotes 1463
\cs_set_protected:Npn 5198		\endtabularnotes 1823
\cs_set_protected_nopar:Npn 4717, 4934		\endVNiceMatrix 5156
D		\endvNiceMatrix 5153
\Ddots 1138, 3301, 3302, 3307, 3308		\enskip 1685, 1707
\ddots 1073, 1130		\ensuremath 3262, 3278, 3294, 3318, 3341
\diagbox 1151, 4717, 4934		\everycr 131, 150, 1111
dim commands:		exp commands:
\dim_add:Nn 4613		\exp_after:wN 198, 1358, 1422, 1440, 1491, 5503
\dim_compare:nNnTF 98, 105, 905, 911, 917, 1525, 2048, 2213, 2730, 2752, 2941, 4526, 4536, 4977, 4997, 5227		\exp_args:Ne 3363
\dim_gzero:N 909, 915, 921		\exp_args:NNc 4451
\dim_max:nn 4515, 4519		\exp_args:NNe 3359
\dim_min:nn 4507, 4511		\exp_args:Nne 2269
\dim_ratio:nn 3000, 3056, 3172, 3177, 3188, 3196, 3205, 3210, 3221, 3229		\exp_args:NNV 1960, 3249, 3265, 3281, 3297, 3321, 3422, 3498, 3576
\dim_set:Nn 4488, 4495, 4506, 4510, 4514, 4518, 4530, 4531, 4540, 4541, 4585, 4598		\exp_args:NNv 1350
\dim_set_eq:NN 4486, 4493, 4593, 4607		\exp_args:NNx 1046, 1729
\dim_sub:Nn 4610		\exp_args:Nnx 4744, 4751, 4826, 4840
\dim_use:N 4527, 4537, 4588, 4589, 4601, 4602, 4625, 4626, 4627, 4628, 4652, 4653, 4654, 4655		\exp_args:No 3073
\dim_zero_new:N 4485, 4487, 4492, 4494		\exp_args:NV ... 1472, 1936, 1963, 1965, 5102
\c_max_dim 2717, 2730, 2739, 2752, 4486, 4488, 4493, 4495, 4527, 4537, 4964, 4977, 4982, 4997, 5315, 5316, 5450, 5451		\exp_args:Nxx 4710, 4711
\l_tmpc_dim 282, 3861, 3862, 3882, 4019, 4027, 4032, 4037, 4043, 4144, 4155, 4160, 4166, 4956, 4962, 5004, 5118, 5129, 5259, 5262, 5270		\exp_last_unbraced:NV 1228, 2425
\l_tmpd_dim 283, 3879, 3882, 4028, 4032, 4151, 4155, 4958, 4962, 4982, 4993, 4997, 5000, 5004, 5127, 5130, 5261, 5262, 5274		\exp_not:N 42, 43, 46, 47, 1569, 1613, 3899, 3910, 4367, 4368, 4927, 5611
\dotfill 1149, 5216		\exp_not:n 981, 2438, 3432, 3508, 3891, 3899, 3901, 3911, 4726, 4802, 4814, 4815, 4916, 4928, 4943, 5237, 5238
\dots 1075		\ExplSyntaxOff . 218, 2314, 2335, 2361, 2440, 4466
\doublerulesep 1620, 4029, 4041, 4152, 4164, 4199		\ExplSyntaxOn . 215, 2300, 2321, 2340, 2433, 4459
\doublerulesepcolor 102		\extrarowheight ... 2847, 4739, 4822, 4835, 5473
		F
		\fi 110, 1474, 4186, 4203
		fi commands:
		\fi: 253
		flag commands:
		\flag_clear_new:n 5602
		\flag_height:n 5629
		\flag_raise:n 5628
		\fontdimen 1786
		fp commands:
		\fp_eval:n 3113
		\fp_to_dim:n 3148
		\futurelet 115
		G
		\globaldefs 1458, 4892
		group commands:
		\group_insert_after:N 5221, 5222, 5224, 5225
		H
		\halign 1125

`\hbox` 1010,
 1426, 1855, 1894, 2013, 2031, 2058, 2062, 2088
 hbox commands:
 `\hbox:n` 64, 66, 69
 `\hbox_gset:Nn` 4731
 `\hbox_overlap_left:n` 1992, 2165
 `\hbox_overlap_right:n` 384, 2109, 2211
 `\hbox_set:Nn`
 367, 1290, 1292, 1417, 1900, 1915, 4946
 `\hbox_set:Nw` 831, 1298, 1657, 2140, 2186
 `\hbox_set_end:` .. 927, 1369, 1665, 2159, 2204
 `\hbox_to_wd:nn` 409, 434
 `\Hdotsfor` 1143, 5802, 5958
 `\hdotsfor` 1076
 `\hdottedline` 1140
 `\heavyrulewidth` 1833
 `\hfil` 1547
 `\hfill` 128, 145
 `\Hline` 1141, 4189
 `\hline` 108
 `\hrule` 112, 128, 145, 1055, 1833, 5349, 5663, 5687
 `\hskip` 111
 `\Hspace` 1142
 `\hspace` 3354
 `\hss` 1547

I

`\ialign` 1007, 1101, 1124, 4871
`\iddots` 1139, 3325, 3326, 3331, 3332
`\iddots` 59, 1074, 1131
 if commands:
 `\if_mode_math:` 251
 `\IfBooleanTF` 1360
 `\ifnum` 110, 4186, 4203
 `\ifstandalone` 1337
 int commands:
 `\int_case:nnTF` 3299, 3305, 3323, 3329
 `\int_compare:nNnTF` 123, 124, 140, 828, 829,
 839, 846, 882, 892, 1052, 1054, 1184, 1186,
 1234, 1244, 1263, 1370, 1374, 1385, 1389,
 1390, 1454, 1684, 1811, 1850, 1889, 1961,
 1989, 2189, 2525, 2532, 2536, 2538, 2593,
 2600, 2604, 2606, 2769, 2808, 2889, 2924,
 2926, 3382, 3396, 3484, 3559, 3823, 3857,
 3874, 3906, 3923, 3924, 3931, 3932, 3936,
 4341, 4343, 4345, 4347, 4735, 4766, 4778,
 5024, 5053, 5057, 5120, 5122, 5176, 5178,
 5180, 5184, 5186, 5188, 5190, 5192, 5507, 5509
 `\int_compare_p:n`
 2671, 2672, 2673, 2674, 3838, 3840, 5112, 5113
 `\int_do_until:nNnn` 3774
 `\int_gadd:Nn` 3395
 `\int_gincr:N` .. 827, 855, 1340, 1593, 1649,
 1671, 1677, 2084, 2184, 2987, 3043, 4444, 4716
 `\int_if_even:nTF` 3726, 5629
 `\int_if_odd_p:n` 3771
 `\int_incr:N` 363, 1603
 `\int_min:nn` 2500, 2502, 2679, 2680
 `\int_step_inline:nn`
 2496, 3722, 3724, 5520, 5537
 `\int_step_inline:nnnn` 4273, 4291, 4306, 4309
 `\l_tmpc_int` 3772, 3773, 3774
 `\c_zero_int` 1684, 3630, 3829

iow commands:

`\iow_now:Nn` 76,
 213, 2340, 2341, 2343, 2361, 2433, 2434, 2440
`\iow_shipout:Nn` 2300, 2301, 2308,
 2314, 2321, 2322, 2329, 2335, 4459, 4460, 4466
`\item` 1816, 1822

K

`\kern` 69
 keys commands:
 `\keys_define:nn`
 476, 496, 503, 572, 619, 671, 710,
 757, 775, 796, 809, 3345, 3733, 4433, 4663,
 4849, 5137, 5279, 5281, 5361, 5376, 5381, 5719
 `\l_keys_key_str`
 5701, 5866, 5872, 5951, 6040,
 6045, 6055, 6070, 6091, 6109, 6152, 6203, 6254
 `\keys_set:nn` 505, 519, 746, 749,
 756, 1354, 1355, 2268, 2278, 2287, 2448,
 2772, 2811, 2892, 2961, 3013, 3487, 3562,
 3579, 3747, 4446, 4908, 5283, 5287, 5404, 5445
 `\keys_set_known:nn` .. 3311, 3335, 4699, 5094
 `\l_keys_value_tl` 5913, 5973, 5980, 6303

L

`\ldots` 1135, 3252, 3255
`\ldots` 1070, 1127
`\leaders` 128, 145
`\left` 1422, 1903, 1918, 5345, 5659, 5683
 legacy commands:
 `\legacy_if:nTF` 603
`\line` 2420, 5934, 6066

M

`\makebox` 1668
`\mathinner` 61
 mode commands:
 `\mode_leave_vertical:` 1329, 1638
 msg commands:
 `\msg_error:nn` 12
 `\msg_error:nnn` 13
 `\msg_error:nnnn` 14, 4889, 4896, 4900
 `\msg_fatal:nn` 15
 `\msg_fatal:nnn` 16
 `\msg_new:nnn` 17
 `\msg_new:nnnn` 18
 `\msg_redirect_name:nnn` 20
`\multicolumn` 1145, 3356, 3408, 3414, 3435
`\multispan` 124, 125, 141, 142
`\myfiledate` 6
`\myfileversion` 7

N

`\newcolumntype` 225, 226, 227
`\newcounter` 323
`\NewDocumentCommand` ... 335, 358, 755, 2447,
 3249, 3265, 3281, 3297, 3321, 3422, 3498,
 3576, 3655, 3670, 3685, 3694, 3715, 3720,
 3742, 3887, 3895, 3904, 5169, 5209, 5411, 5424
`\NewDocumentEnvironment` 1312,
 1933, 1942, 2224, 2234, 2264, 2275, 2283, 4442
`\NewExpandableDocumentCommand` 231, 4674
`\newlist` 339, 350
`\NiceArray` 2280, 2289
`\NiceArrayWithDelims` 2229, 2239

nicematrix commands:	
\g_nicematrix_code_after_tl	
267, 612, 1957, 2425, 2427, 4913, 5294, 5301	
\g_nicematrix_code_before_tl	
... 1169, 2429, 2438, 3889, 3897, 3908, 4925	
\NiceMatrixLastEnv	231
\NiceMatrixOptions	755, 6110, 6308
\NiceMatrixoptions	5977
\noalign 98, 105, 110, 133, 1034, 1107, 4186, 4355	
\nobreak	353
\normalbaselines	1095
\NotEmpty	1152
\nulldelimiterspace	1914, 1926, 5330, 5499
\nullfont	5341, 5348, 5655, 5662, 5679, 5686
\numexpr	153, 154
O	
\omit	123, 1991, 2007, 2083, 5291
\OnlyMainNiceMatrix	1148, 3915
P	
\par	1810, 1818
peek commands:	
\peek_meaning:NTF	158, 365, 1064
\peek_meaning_ignore_spaces:NTF	1935, 4189
\peek_meaning_remove_ignore_spaces:NTF	148
\peek_remove_spaces:n	3380
\pgfextracty	5027
\pgfgetlastxy	427
\pgfpathcircle	3237
\pgfpathlineto	
4037, 4043, 4160, 4166, 5262, 5514, 5531, 5565	
\pgfpathmoveto	
4036, 4042, 4159, 4165, 5257, 5513, 5530, 5556	
\pgfpathrectanglecorners	3881, 4030, 4153, 5128
\pgfpointadd	425
\pgfpointanchor	163,
234, 2695, 2706, 2723, 2745, 2853, 2871,	
4505, 4513, 4972, 4990, 5009, 5011, 5028,	
5062, 5322, 5459, 5466, 5482, 5489, 5601, 5607	
\pgfpointdiff	426, 1197, 1205
\pgfpointlineattime	3107
\pgfpointorigin	1998, 2121
\pgfpointscale	425
\pgfpointshapeborder	3617, 3620
\pgfrememberpicturepositiononpagetrue ...	
... 860, 953, 1022, 1997,	
2021, 2039, 2070, 2096, 2119, 2367, 2480,	
2499, 3064, 3616, 4012, 4137, 4376, 4421,	
4548, 4558, 4569, 4949, 5096, 5252, 5308, 5447	
\pgfscope	3104, 5269
\pgfset 394, 419, 954, 5039, 5073, 5268, 5329, 5449	
\pgfsetbaseline	952
\pgfsetcornersarced	3880, 5104
\pgfsetlinewidth	4046, 4169, 5131, 5501
\pgfsetrectcap	4047, 4170
\pgfsetroundcap	5265
\pgfsetstrokecolor	5102
\pgfsyspdfmark	216, 217
\pgftransformrotate	3111
\pgftransformshift	400, 425,
3105, 5038, 5060, 5270, 5274, 5331, 5579, 5589	
\pgfusepath	
... 3131, 3141, 3646, 3799, 3811, 4033, 5132	
\pgfusepathqfill	3243, 4156
\pgfusepathqstroke	
... 4048, 4171, 5266, 5515, 5532, 5566	
\phantom	3262, 3278, 3294, 3318, 3341
\pNiceMatrix	5149
prg commands:	
\prg_do_nothing:	
... 177, 186, 192, 220, 493, 1107, 2423	
\prg_new_conditional:Nnn	3826, 3834
\prg_replicate:nn	373, 2079,
2080, 3435, 4038, 4161, 5179, 5182, 5185, 5191	
\prg_return_false:	3831, 3843
\prg_return_true:	3832, 3842
\ProcessKeysOptions	5738
\ProvideDocumentCommand	59
\ProvidesExplPackage	4
Q	
\quad	354
quark commands:	
\q_stop	120,
121, 137, 138, 159, 161, 1358, 1491, 1558,	
1705, 1952, 1955, 3570, 3584, 3585, 3650,	
3705, 3710, 3766, 3852, 3853, 3869, 3870,	
4642, 4649, 4676, 4679, 5110, 5119, 5164,	
5173, 5413, 5418, 5431, 5434, 5503, 5614, 5622	
R	
\rectanglecolor	1219, 2431, 3899
\refstepcounter	382
regex commands:	
\regex_const:Nn	210
\regex_match:nnTF	5393
\regex_replace_all:NnN	1496
\relax	153, 154
\renewcommand	196
\RenewDocumentEnvironment	
... 5148, 5151, 5154, 5157, 5160	
\RequirePackage	1, 3, 9, 10, 11
\right	1440, 1910, 1922, 5354, 5667, 5691
\rotated	1147
\roundedrectanglecolor	1220, 4927
\rowcolor	1081, 1221
\rowcolors	1222
S	
\savenotes	1316
scan commands:	
\scan_stop:	2426
\scriptstyle	
... 835, 2142, 2188, 3092, 3093, 3127, 3137	
seq commands:	
\seq_clear:N	1489
\seq_clear_new:N	1227, 2342, 4251
\seq_count:N	1962, 3633
\seq_gclear:N	
... 1173, 1343, 1344, 1345, 1346, 1838, 2424	
\seq_gclear_new:N ...	1155, 1156, 1958, 1974
\seq_gpop_left:NN	1964, 1976
\seq_gput_left:Nn	608, 3384, 3386, 4906
\seq_gput_right:Nn	364, 1566,
2650, 3387, 4799, 4811, 4920, 5240, 5399, 5419	
\seq_gset_from_clist:Nn	2345, 2357
\seq_gset_map_x:NNn	2451

`\seq_gset_split:Nnn` 1960, 1975
`\seq_if_empty:NTF` 1798
`\seq_if_empty_p:N` 2397, 2398, 2399
`\seq_if_exist:NTF` 1176
`\seq_if_in:NnTF`
 606, 3989, 3995, 4002, 4114,
 4120, 4127, 4508, 4516, 4970, 4988, 5395, 5788
`\seq_item:Nn` 1180, 1183, 1192, 1193, 1200, 1201
`\seq_map_function:NN` 1966
`\seq_map_indexed_inline:Nn` 3628, 3642
`\seq_map_inline:Nn`
 1816, 1822, 1978, 2664, 3786, 3950,
 3952, 3954, 4076, 4078, 4080, 4336, 4872, 5505
`\seq_mapthread_function:NNN` 4637
`\seq_new:N` 242,
 256, 287, 288, 289, 290, 291, 296, 324, 5778
`\seq_put_right:Nn` 3632, 4323
`\seq_set_eq:NN` 3755
`\seq_set_filter:NNn` 3757, 3784
`\seq_set_from_clist:Nn` 5779
`\seq_set_map_x:NNn` 5784
`\seq_use:Nnnn` 2359, 5947, 6313
`\l_tmpa_seq` 3757, 3784
`\l_tmpb_seq` 3755, 3757, 3784, 3786
`\setlist` 340, 351, 644, 649, 660, 665
skip commands:
`\skip_gadd:Nn` 2055
`\skip_gset:Nn` 2046
`\skip_gset_eq:NN` 2053, 2054
`\skip_horizontal:N` 129, 146, 1299,
 1300, 1367, 1368, 1392, 1393, 1429, 1430,
 1433, 1434, 1449, 1450, 1480, 1486, 1569,
 1713, 1726, 1733, 1927, 1928, 1930, 1931,
 2002, 2003, 2015, 2017, 2033, 2035, 2057,
 2064, 2066, 2085, 2090, 2092, 2111, 2112,
 2170, 2171, 2172, 2175, 2210, 2215, 2216, 2217
`\skip_horizontal:n` 385, 1615
`\skip_vertical:N`
 133, 1016, 1018, 1425, 1436, 1807, 1832, 4355
`\skip_vertical:n` 897, 4196
`\g_tmpa_skip` 2046, 2053, 2054, 2055, 2057, 2085
`\c_zero_skip` 1112
`\space` 264, 265
`\stepcounter` 371, 376
str commands:
`\c_backslash_str` 264
`\c_colon_str` 754
`\str_case:nn`
 ... 3363, 5031, 5043, 5065, 5077, 5550, 5559
`\str_case:nnTF`
 1397, 1534, 1772, 4254, 5626, 5639
`\str_clear_new:N` 5442
`\str_foldcase:n` 3363
`\str_gclear:N` 2443
`\str_gset:Nn`
 ... 1326, 2228, 2237, 2266, 2277, 2285, 5200
`\str_if_empty:NTF`
 . 864, 965, 1025, 1247, 1265, 1325, 1999,
 2024, 2042, 2073, 2099, 2122, 2227, 2236,
 2306, 2327, 2364, 4629, 4656, 5571, 5585, 5595
`\str_if_eq:nnTF` 84, 263, 1005, 1561,
 1564, 1608, 1720, 1947, 1949, 1982, 5100, 5299

`\str_if_eq_p:nn`
 482, 1574, 1702, 1703, 1704, 1705, 4684, 4689
`\str_if_in:NnTF` 1760, 1864
`\str_new:N` 259, 464, 753
`\str_range:Nnn` 1764, 1868
`\str_set:Nn` 605, 742, 5398
`\str_set_eq:NN` 609, 754
`\l_tmpa_str` 605, 606, 608, 609
`\strut` 1816, 1822
`\strutbox` 2847, 2865, 5473, 5476
`\SubMatrix`
 1225, 2407, 5422, 5882, 5914, 5921, 5940, 6073

T

`\tabcolsep` 998, 1429, 1433
`\tabskip` 1112
`\tabularnote` 335, 358, 365, 6005, 6018
`\tabularnotes` 1821
TeX and L^AT_EX 2_ε commands:
`\@BTnormal` 1087
`\@acol` 988
`\@acoll` 986
`\@acolr` 987
`\@array@array` 990
`\@arrayacol` 986, 987, 988
`\@arstrutbox` 857, 858,
 897, 1114, 1116, 1118, 1121, 1123, 1640, 1644
`\@currenvir` 5299
`\@depth` 5351, 5664, 5688
`\@gobblethree` 217
`\@halignto` 989, 1001, 1002
`\@height` 113, 128, 145, 5349, 5663, 5687
`\@ifclassloaded` 51, 54, 5759, 5769
`\@ifnextchar` 1160
`\@ifpackageloaded`
 30, 33, 36, 39, 75, 91, 167, 5762, 5772
`\@mainaux` 76, 213, 2300, 2301, 2308,
 2314, 2321, 2322, 2329, 2335, 2340, 2341,
 2343, 2361, 2433, 2434, 2440, 4459, 4460, 4466
`\@tabarray` 1003
`\@tempswafalse` 1474
`\@tempswatruer` 1473
`\@temptokena` .. 176, 179, 198, 200, 1472, 1491
`\@whiles` 1474
`\@width` 113, 5352, 5665, 5689
`\@xhline` 116
`\bBigg@` 1290, 1292
`\c@MaxMatrixCols` 2256, 5816
`\c@tabularnote` 1800, 1811, 1839
`\col@sep` 998, 999, 1392,
 1450, 2002, 2055, 2111, 2175, 2210, 2734, 2756
`\CT@arc` 95, 96
`\CT@arc@`
 ... 94, 99, 114, 127, 144, 160, 162, 1333,
 1833, 2445, 4045, 4168, 4420, 5101, 5264, 5504
`\CT@dr` 102, 103
`\CT@drsc@` .. 101, 106, 4022, 4025, 4147, 4150
`\CT@everycr` 1105
`\CT@row@color` 1107
`\if@temp` 1474
`\NC@` 1063
`\NC@find` 177, 205
`\NC@list` 1474
`\NC@rewrite@S` 178, 196

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>	7
5.2	The thickness and the color of the rules	8
5.3	The keys <code>hlines</code> and <code>vlines</code>	8
5.4	The key <code>hvlines</code>	8
5.5	The key <code>hvlines-except-corners</code>	9
5.6	The command <code>\diagbox</code>	9
5.7	Dotted rules	10
6	The color of the rows and columns	10
6.1	Use of <code>colortbl</code>	10
6.2	The tools of <code>nicematrix</code> in the code-before	11
6.3	Color tools with the syntax of <code>colortbl</code>	13
7	The width of the columns	14
8	The exterior rows and columns	15
9	The continuous dotted lines	17
9.1	The option <code>nullify-dots</code>	18
9.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	19
9.3	How to generate the continuous dotted lines transparently	20
9.4	The labels of the dotted lines	20
9.5	Customisation of the dotted lines	20
9.6	The dotted lines and the rules	21
10	The <code>\CodeAfter</code>	22
10.1	The command <code>\line</code> in the <code>\CodeAfter</code>	22
10.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	22
11	The notes in the tabulars	24
11.1	The footnotes	24
11.2	The notes of tabular	24
11.3	Customisation of the tabular notes	25
11.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	28

12	Other features	28
12.1	Use of the column type S of siunitx	28
12.2	Alignment option in {NiceMatrix}	28
12.3	The command \rotate	28
12.4	The option small	29
12.5	The counters iRow and jCol	29
12.6	The option light-syntax	30
12.7	Color of the delimiters	30
12.8	The environment {NiceArrayWithDelims}	31
13	Use of Tikz with nicematrix	31
13.1	The nodes corresponding to the contents of the cells	31
13.2	The “medium nodes” and the “large nodes”	32
13.3	The nodes which indicate the position of the rules	33
13.4	The nodes corresponding to the command \SubMatrix	34
14	API for the developpers	34
15	Technical remarks	35
15.1	Definition of new column types	35
15.2	Diagonal lines	35
15.3	The “empty” cells	36
15.4	The option exterior-arraycolsep	36
15.5	Incompatibilities	37
16	Examples	37
16.1	Notes in the tabulars	37
16.2	Dotted lines	38
16.3	Dotted lines which are no longer dotted	39
16.4	Stacks of matrices	40
16.5	How to highlight cells of a matrix	43
16.6	Utilisation of \SubMatrix in the code-before	45
17	Implementation	45
18	History	188
	Index	194