

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 18, 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 such as MiKTeX, TeXLive or MacTeX.

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

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

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

If you use TeXLive as TeX distribution, you should note that TeXLive 2020 at least is required by `nicematrix`.

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

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

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

*This document corresponds to the version 6.3 of `nicematrix`, at the date of 2021/10/18.

¹The latest version of the file `nicematrix.sty` may be downloaded from the SVN server of TeXLive:

<https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty>

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

1 The environments of this package

The package `nicematrix` defines the following new environments.

<code>{NiceTabular}</code>	<code>{NiceArray}</code>	<code>{NiceMatrix}</code>
<code>{NiceTabular*}</code>	<code>{pNiceArray}</code>	<code>{pNiceMatrix}</code>
<code>{NiceTabularX}</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.

The environment `{NiceTabularX}` is similar to the environment `{tabularx}` from the eponymous package.³

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

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

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`.⁴

```

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


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


```

³In fact, it's possible to use directly the `X` columns in the environment `{NiceTabular}` (and the required width for the tabular is fixed by the key `width`): cf. p. 19

⁴One 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`⁵: `\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}

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

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

⁵The extension `booktabs` is *not* loaded by `nicematrix`.

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.⁶

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}`, `{NiceTabular*}` and `{NiceTabularX}`, 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.

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

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

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.⁷

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

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

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

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

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

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:

⁶The spaces after a command `\Block` are deleted.

⁷This 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 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;
- 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` or the key `hvlines` is in force);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt⁸);
- the key `borders` provides the ability to draw only some borders of the blocks; the value of that key is a (comma-separated) list of elements covered by `left`, `right`, `top` and `bottom`;
- the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`);
- the keys `hvlines` draws all the vertical and horizontal rules in the block;
- when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz⁹ by using as options the value of that key `tikz` (which must be a list of keys allowed for a Tikz path). For examples, cf. p. 44;
- **New 6.3** the key `name` provides a name to the rectangular Tikz node corresponding to the block.

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      & tulip & daisy & dahlia \\
violet    &        &        &        \\
& \Block[draw=red,fill=[RGB]{204,204,255},rounded-corners]{2-2}
&        &        &        \\
& & & marigold \\
iris & & & lis \\
arum & periwinkle & forget-me-not & hyacinth
\end{NiceTabular}
```

rose	tulip	daisy	dahlia
violet	Some beautiful flowers		
iris			marigold
arum	periwinkle	forget-me-not	lis
			hyacinth

⁸This value is the initial value of the *rounded corners* of Tikz.

⁹Tikz should be loaded (by default, `nicematrix` only loads `PGF`) and, if it's not, an error will be raised.

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.

New 6.0 In the columns with a fixed width (columns `w{...}{...}`, `p{...}`, `b{...}`, `m{...}` and `X`), the content of the block is formatted as a paragraph of that width.

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

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

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

4.3 The mono-row blocks

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

4.4 The mono-cell blocks

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

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

- It's possible to use the command `\` in a (mono-cell) block.
- It's possible to use the option of horizontal alignment of the block in derogation of the type of column given in the preamble of the array.
- It's possible to draw a frame around the cell with the key `draw` of the command `\Block` and to fill the background with rounded corners with the keys `fill` and `rounded-corners`.¹⁰
- It's possible to draw one or several borders of the cell with the key `borders`.

¹⁰If one simply wishes to color the background of a unique cell, there is no point using the command `\Block`: it's possible to use the command `\cellcolor` (when the key `colortbl-like` is used).

```

\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.¹¹

4.5 Horizontal position of the content of the block

By default, the horizontal position of the content of a block is computed by using the positions of the *contents* of the columns implied in that block. That's why, in the following example, the header “First group” is correctly centered despite the instruction `!\qquad` in the preamble which has been used to increase the space between the columns (this is not the behaviour of `\multicolumn`).

```

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

```

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

In order to have an horizontal positionning of the content of the block computed with the limits of the columns of the LaTeX array (and not with the contents of those columns), one may use the key `L`, `R` and `C` of the command `\Block`.

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.

¹¹One may consider that the default value of the first mandatory argument of `\Block` is `1-1`.

5.1 Some differences with the classical environments

5.1.1 The vertical rules

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

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

First	Second
Peter	
Mary	George

However, the vertical rules are not drawn in the blocks (created by `\Block`: cf. p. 4) nor in the corners (created by the key `corner`: cf. p. 10).

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. 42):

```
\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	B	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}
A&B&C&D \\ \hline
\end{NiceTabular}
```

A	B	C	D
A	B	C	D

New 6.2

In the environments of `nicematrix`, an instruction `\cline{i}` is equivalent to `\cline{i-i}`.

5.2 The thickness and the color of the rules

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

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

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

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

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

If one wishes to define new specifiers for columns in order to draw vertical rules (for example with a specific color or thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 42.

5.3 The tools of `nicematrix` for the rules

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

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

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

- These blocks are:
 - the blocks created by the command `\Block`¹² presented p. 4;
 - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `\Vdots`, etc. (cf. p. 21).
- The corners are created by the key `corners` explained below (see p. 10).

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

5.3.1 The keys `hlines` and `vlines`

The keys `hlines` and `vlines` (which draw, of course, horizontal and vertical rules) take in as value a list of numbers which are the numbers of the rules to draw.¹³

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

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

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

¹²And also the command `\multicolumn` but it's recommended to use instead `\Block` in the environments of `nicematrix`.

¹³It's possible to put in that list some intervals of integers with the syntax `i-j`.

5.3.2 The keys hvlines and hvlines-except-borders

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

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

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

The key `hvlines-except-borders` is similar to the key `hvlines` but does not draw the rules on the horizontal and vertical borders of the array.

5.3.3 The (empty) corners

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

For each of these corners, we will call *empty corner* (or simply *corner*) the reunion of all the empty rectangles starting from the cell actually in the corner of the array.¹⁴

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

				A	
			A	A	
				A	
			A	A	A
A	A	A	A	A	A
A	A	A	A	A	A
		A	A		
			A		
			A		

In the example on the right (where B is in the center of a block of size 2×2), we have colored in blue the four (empty) corners of the array.

When the key `corners` is used, `nicematrix` computes the (empty) corners and these corners will be taken into account by the tools for drawing the rules (the rules won't be drawn in the corners).

Remark: In the previous versions of `nicematrix`, there was only a key `hvlines-except-corners` (now considered as obsolete).

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

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

¹⁴For sake of completeness, we should also say that a cell contained in a block (even an empty cell) is not taken into account for the determination of the corners. That behaviour is natural.

It's also possible to provide to the key `corners` a (comma-separated) list of corners (designed by NW, SW, NE and SE).

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

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

▷ The corners are also taken into account by the tools provided by `nicematrix` to color cells, rows and columns. These tools don't color the cells which are in the corners (cf. p. 12).

5.4 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.¹⁵

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

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

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

5.5 Dotted rules

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

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

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

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`¹⁶).

¹⁵The author of this document considers that type of construction as graphically poor.

¹⁶However, one should remark that the package `arydshln` is not fully compatible with `nicematrix`.

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 rule¹⁷. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

6 The color of the rows and columns

6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

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

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

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

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

The package `nicematrix` provides some tools (independent of `colortbl`) to draw the colored panels first, and, then, the content of the cells and the rules. This strategy is more conform to the “painting model” of the formats PostScript and PDF and is more suitable for the PDF viewers. However, it requires several compilations.¹⁸

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

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

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

¹⁷In fact, with `array`, this is true only for `\hline` and “|” but not for `\cline`: cf p. 8

¹⁸If you use Overleaf, Overleaf will do automatically the right number of compilations.

New commands are available in that `\CodeBefore`: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors`, `\rowlistcolors`, `\chessboardcolors` and `arraycolor`.¹⁹

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.

These commands don't color the cells which are in the “corners” if the key `corners` is used. This key has been described p. 10.

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

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

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

a	b	c
e	f	g
h	i	j

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

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

a	b	c
e	f	g
h	i	j

- The command `\arraycolor` takes in as mandatory argument a color and color the whole tabular with that color (excepted the potential exterior rows and columns: cf. p. 20). It's only a particular case of `\rectanglecolor`.
- The command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

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

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

We have used the key `r` which aligns all the columns rightwards (cf. p. 34).

¹⁹Remark that, in the `\CodeBefore`, PGF/Tikz nodes of the form “(i-lj)” are also available to indicate the position to the potential rules: cf. p. 40.

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

```

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

```

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

- The command `\columncolor` takes its name from the command `\columncolor` of `colortbl`. Its syntax is similar to the syntax of `\rowcolor`.
- The command `\rowcolors` (with a s) takes its name from the command `\rowcolors` of `xcolor`²⁰. 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 $i-j$ (where i or j may be replaced by $*$).
- With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.²¹
- 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).

²⁰The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

²¹Otherwise, the color of a given row relies only upon the parity of its absolute number.

```

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

```

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

```

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

```

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

- **New 6.0** The extension `nicematrix` provides also a command `\rowlistcolors`. This command generalises the command `\rowcolors`: instead of two successive arguments for the colors, this command takes in an argument which is a (comma-separated) list of colors. In that list, the symbol `=` represent a color identical to the previous one.

```

\begin{NiceTabular}{c}
\CodeBefore
  \rowlistcolors{1}{red!15,blue!15,green!15}
\Body
Peter \\\
James \\\
Abigail \\\
Elisabeth \\\
Claudius \\\
Jane \\\
Alexandra \\\
\end{NiceTabular}

```

Peter
James
Abigail
Elisabeth
Claudius
Jane
Alexandra

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

```

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

```

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

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

```

\begin{NiceTabular}[c]{1SSSS}
\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{rl}{2-4}
& L & l & h & \\
\midrule
small & 3 & 5.5 & 1 & 30 \\
standard & 5.5 & 8 & 1.5 & 50.5 \\
premium & 8.5 & 10.5 & 2 & 80 \\
extra & 8.5 & 10 & 1.5 & 85.5 \\
special & 12 & 12 & 0.5 & 70 \\
\bottomrule
\end{NiceTabular}

```

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

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

6.3 Color tools with the syntax of `colortbl`

It's possible to access the preceding tools with a syntax close to the syntax of `colortbl`. For that, one must use the key `colortbl-like` in the current environment.²²

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

```

²²Up to now, this key is *not* available in `\NiceMatrixOptions`.


```

\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 command `\RowStyle`

The command `\RowStyle` takes in as argument some formatting intructions that will be applied to each cell on the rest of the current row.

That command also takes in as optional argument (between square brackets) a list of key-value pairs.

- **New 6.3** The key `nb-rows` sets the number of rows to which the specifications of the current command will apply.
- The keys `cell-space-top-limit`, `cell-space-bottom-limit` and `cell-space-limits` are available with the same meaning that the corresponding global keys (cf. p. 2).
- **New 6.3** The key `rowcolor` sets the color of the background and the key `color` sets the color of the text.²³
- **New 6.3** The key `bold` enforces bold characters for the cells of the row, both in math mode and text mode.

```

\begin{NiceTabular}{cccc}
\hline
\RowStyle[cell-space-limits=3pt]{\rotate}
first & second & third & fourth \\
\RowStyle[nb-rows=2,rowcolor=blue!50,color=white]{\sffamily}
1 & 2 & 3 & 4 \\
I & II & III & IV
\end{NiceTabular}

```

first	second	third	fourth
-------	--------	-------	--------

The command `\rotate` is described p. 34.

8 The width of the columns

8.1 Basic tools

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`, `W`, `p`, `b` and `m` of the package `array`.

²³The key `color` uses the command `\color` but inserts also an instruction `\leavevmode` before. This instruction prevents a extra vertical space in the cells which belong to columns of type `p`, `b`, `m` and `X` (which start in vertical mode).

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

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

In the environments of `nicematrix`, it's also possible to fix the *minimal* width of all the columns (excepted the potential exterior columns: cf. p. 20) 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.²⁴

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

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

Without surprise, it's possible to fix the minimal width of the columns of all the arrays 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`²⁵. 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}
```

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

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

²⁴The 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).

²⁵At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

8.2 The columns V of varwidth

New 6.3

Let's recall first the behaviour of the environment `{varwidth}` of the eponymous package `varwidth`. That environment is similar to the classical environment `{minipage}` but the width provided in the argument is only the *maximal* width of the created box. In the general case, the width of the box constructed by an environment `{varwidth}` is the natural width of its contents.

That point is illustrated on the following examples.

```
\fbox{%
\begin{varwidth}{8cm}
\begin{itemize}
\item first item
\item second item
\end{itemize}
\end{varwidth}}
```

- first item
 - second item

```
\fbox{%
\begin{minipage}{8cm}
\begin{itemize}
\item first item
\item second item
\end{itemize}
\end{minipage}}
```

- first item
 - second item

The package `varwidth` provides also the column type `V`. A column of type `V{<dim>}` encapsulates all its cells in a `{varwidth}` with the argument `<dim>` (and does also some tuning).

When the package `varwidth` is loaded, the columns `V` of `varwidth` are supported by `nicematrix`. Concerning `nicematrix`, one of the interests of this type of columns is that, for a cell of a column of type `V`, the PGF/Tikz node created by `nicematrix` for the content of that cell has a width adjusted to the content of the cell : cf. p. 38.

8.3 The columns X

New 6.0

The environment `{NiceTabular}` provides `X` columns similar to those provided by the environment `{tabularx}` of the eponymous package.

The required width of the tabular may be specified with the key `width` (in `{NiceTabular}` or in `\NiceMatrixOptions`). The initial value of this parameter is `\linewidth` (and not `\textwidth`).

For sake of similarity with the environment `{tabularx}`, `nicematrix` also provides an environment `{NiceTabularX}` with a first mandatory argument which is the width of the tabular.²⁶

As with the packages `tabu` and `tabularray`, the specifier `X` takes in an optional argument (between square brackets) which is a list of keys.

- It's possible to give a weight for the column by providing a positive integer directly as argument of the specifier `X`. For example, a column `X[2]` will have a width double of the width of a column `X` (which has a weight equal to 1).²⁷
- It's possible to specify an horizontal alignment with one of the letters `l`, `c` and `r` (which insert respectively `\raggedright`, `\centering` and `\raggedleft` followed by `\arraybackslash`).
- It's possible to specify a vertical alignment with one of the keys `t` (alias `p`), `m` and `b` (which construct respectively columns of type `p`, `m` and `b`). The default value is `t`.

²⁶If `tabularx` is loaded, one must use `{NiceTabularX}` (and not `{NiceTabular}`) in order to use the columns `X` (this point comes from a conflict in the definitions of the specifier `X`).

²⁷The negative values of the weight, as provided by `tabu` (which is now obsolete), are not supported by `nicematrix`. If such a value is used, an error will be raised.

```
\begin{NiceTabular}[width=9cm]{X[2,1]X[1]}[hvlines]
a rather long text which fits on several lines
& a rather long text which fits on several lines \\
a shorter text & a shorter text
\end{NiceTabular}
```

a rather long text which fits on several lines	a rather long text which fits on several lines
a shorter text	a shorter text

9 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`. It's particularly interesting for the (mathematical) matrices.

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

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

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

The dotted lines have been drawn with the tools presented p. 21.

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.²⁸
- 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. 36) 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).

²⁸The users wishing exteriors columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 27).

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

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

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

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

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

Remarks

- As shown in the previous example, the horizontal and vertical rules don'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 42.
- A specification of color present in `code-for-first-row` also applies to a dotted line drawn in that 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. 17) 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. 27.

10 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`.²⁹

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells³⁰ 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`.³¹

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

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

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

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

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

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

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

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

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

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

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.³²

²⁹The 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`.

³⁰The precise definition of a “non-empty cell” is given below (cf. p. 43).

³¹It's also possible to change the color of all these dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 25.

³²In `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. 17

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

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

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

10.1 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.

10.2 The commands `\Hdotsfor` and `\Vdotsfor`

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

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

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

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

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

```


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


```

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`³³ is loaded (but you might have problem if you use `\rowcolor` on the same row as `\Hdotsfor`).

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

```

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

```

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

10.3 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`.³⁴

³³We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

³⁴The options `renew-dots`, `renew-matrix` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage`. There is also a key `transparent` which is an alias for the conjunction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`²⁹ and `\hdotsfor` are redefined within the environments provided by `nicematrix` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` (“automatic dots” of `amsmath`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

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

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

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

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

10.4 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. 27) accept two optional arguments specified by the tokens `_` and `^` for labels positionned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

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

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

10.5 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. 27) may be customized by three options (specified between square brackets after the command):

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

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

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

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

The option `xdots/color`

The option `xdots/color` fixes the color of the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 20.

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).³⁵

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

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

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

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

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

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

10.6 The dotted lines and the rules

The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier `|` in the preamble, by the command `\Hline` and by the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` are not drawn within the blocks).³⁶

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

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

³⁵The 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.

³⁶On the other side, the command `\line` in the `\CodeAfter` (cf. p. 27) does *not* create block.

11 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.³⁷

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

The experienced users may, for instance, use the PGF/Tikz nodes created by `nicematrix` in the `\CodeAfter`. These nodes are described further beginning on p. 37.

Moreover, two special commands are available in the `\CodeAfter`: `\line` and `\SubMatrix`. We will now present these commands.

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

The command `\line` draws directly dotted lines between nodes. It takes in two arguments for the two cells to link, both of the form i - j where i is the number of the row and j is the number of the column. The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 25).

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

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

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

It can also be used to draw a diagonal line not parallel to the other diagonal lines (by default, the dotted lines drawn by `\Ddots` are “parallelized”: cf. p. 42).

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

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

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

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

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

³⁷There is also a key `code-before` described p. 13.

- 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.³⁸

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

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

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

In fact, the command `\SubMatrix` also takes in two optional arguments specified by the traditional symbols `^` and `_` for material in superscript and subscript.

```
$\begin{bNiceMatrix}[right-margin=1em]
1 & 1 & 1 \\
1 & a & b \\
1 & c & d \\
\CodeAfter
\SubMatrix[{2-2}{3-3}]^T
\end{bNiceMatrix}$
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & a & b \\ 1 & c & d \end{bmatrix}^T$$

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

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

³⁸There 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}]`).

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

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

```


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


```

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

```


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


```

There is also a key `name` which gives a name to the submatrix created by `\SubMatrix`. That name is used to create PGF/Tikz nodes: cf p. 41.

It's also possible to specify some delimiters³⁹ by placing them in the preamble of the environment (for the environments with a preamble: `{NiceArray}`, `{pNiceArray}`, etc.). This syntax is inspired by the extension `blkarray`.

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

```


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


```

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

³⁹Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

12 The notes in the tabulars

12.1 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`.

12.2 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@{}}
\toprule \RowStyle{\bfseries}
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. 31. This table has been composed with the following code.

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

Table 1: Use of `\tabularnote`^a

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

Some text before the notes.

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

^b Considered as the first nurse of history.

^c Nicknamed “the Lady with the Lamp”.

^d The label of the note is overlapping.

12.3 Customisation of the tabular notes

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

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

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

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

We detail these keys.

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

Initial value: `false`

That key is also available within a given environment.

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

Initial value: `false`

That key is also available within a given environment.

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

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

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

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

Initial value: `#1`

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

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

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

Initial value: `#1`

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

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

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

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

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

This initial value contains the specification `align = left` which requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 31).

- 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. 45.

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

If you wish to use the environment `{NiceTabular}`, `{NiceTabular*}` `{NiceTabularX}` 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*}\TPT@hookin{NiceTabularX} }
\makeatother
```

13 Other features

13.1 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`.

```

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

```

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

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

13.2 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).

```

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

```

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

13.3 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`.⁴⁰

```

\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 \backslash\backslash
1 \& 2 \& 3 \& e_1 \backslash\backslash
4 \& 5 \& 6 \& e_2 \backslash\backslash
7 \& 8 \& 9 \& e_3
\end{pNiceMatrix}

```

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

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 \backslash\backslash
4 \& 5 \& 6 \& e_2 \backslash\backslash
7 \& 8 \& 9 \& e_3 \backslash\backslash
\text{image of } e_1 \& e_2 \& e_3
\end{pNiceMatrix}

```

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

⁴⁰It can also be used in `\RowStyle` (cf. p. 17).

13.4 The option `small`

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

```

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

```

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

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

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

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

13.5 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 column⁴¹. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 13) and in the `\CodeAfter` (cf. p. 27), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

```

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

```

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

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

⁴¹We 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.

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).

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

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

13.6 The option `light-syntax`

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

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

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

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

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

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.⁴²

13.7 Color of the delimiters

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

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

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

This colour also applies to the delimiters drawn by the command `\SubMatrix` (cf. p. 27).

13.8 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.

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

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

⁴²The 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.

14 Use of Tikz with nicematrix

14.1 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`. ⁴³

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

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

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

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

```

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

```

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

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

```

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

```

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 51).

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

New 6.3 The nodes of the last column (excepted the potential «last column» specified by `last-col`) may also be indicated by i -last. Similarly, the nodes of the last row may be indicated by last- j .

⁴³One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 21) and the computation of the “corners” (cf. p. 10).

14.1.1 Les colonnes V de varwidth

When the extension `varwidth` is loaded, the columns of the type `V` defined by `varwidth` are supported by `nicematrix`. It may be interesting to notice that, for a cell of a column of type `V`, the PGF/Tikz node created by `nicematrix` for the content of that cell has a width adjusted to the content of the cell. This is in contrast to the case of the columns of type `p`, `m` or `b` for which the nodes have always a width equal to the width of the column.

```
\begin{NiceTabular}{V{8cm}}
\bfseries \large
Titre \\
\lipsum[1]
\CodeAfter
\tikz \draw [rounded corners] (1-1) -| (last-|2) -- (last-|1) |- (1-1) ;
\end{NiceTabular}
```

Titre

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

We have used the nodes corresponding to the position of the potential rules, which are described below (cf. p. 40).

14.2 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`.⁴⁴

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.

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

⁴⁴There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.⁴⁵

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

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.⁴⁶

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

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

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

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

Here is an array composed with the following code:

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

The nodes we have described are not available by default in the `\CodeBefore` (described p. 13). It’s possible to have these nodes available in the `\CodeBefore` by using the key `create-cell-nodes` of the keyword `\CodeBefore` (in that case, the nodes are created first before the construction of the

⁴⁵There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 20).

⁴⁶The 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.

array by using informations written on the **aux** file and created a second time during the contruction of the array itself).

14.3 The nodes which indicate the position of the rules

The package **nicematrix** creates a PGF/Tikz node merely called *i* (with the classical prefix) at the intersection of the horizontal rule of number *i* and the vertical rule of number *i* (more specifically the potential position of those rules because maybe there are not actually drawn). The last node has also an alias called **last**. There is also a node called *i.5* midway between the node *i* and the node *i + 1*. These nodes are available in the **\CodeBefore** and the **\CodeAfter**.

	tulipe	lys
arum		violette mauve
muguet	dahlia	

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

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

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

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

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

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

14.4 The nodes corresponding to the command `\SubMatrix`

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 27.

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}\}`).

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

15 API for the developpers

The package `nicematrix` provides two variables which are internal but public⁴⁷:

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

These variables contain the code of what we have called the “code-before” (usually specified at the beginning of the environment with the syntax using the keywords `\CodeBefore` and `\Body`) and the “code-after” (usually specified at the end of the environment after the keyword `\CodeAfter`). 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 `\crossbox` to draw a cross in the current cell. This command will take in an optional argument between square brackets for a list of pairs *key-value* which will be given to Tikz before the drawing.

It’s possible to program such command `\crossbox` as follows, explicitly using the public variable `\g_nicematrix_code_after_tl`.

```
\ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_crossbox:nnn
{
  \tikz \draw [ #3 ]
    ( #1 -| \int_eval:n { #2 + 1 } ) -- ( \int_eval:n { #1 + 1 } -| #2 )
    ( #1 -| #2 ) -- ( \int_eval:n { #1 + 1 } -| \int_eval:n { #2 + 1 } ) ;
}

\NewDocumentCommand \crossbox { ! 0 { } }
{
  \tl_gput_right:Nx \g_nicematrix_code_after_tl
  {
    \__pantigny_crossbox:nnn
    { \int_use:c { c@iRow } }
    { \int_use:c { c@jCol } }
  }
}
```

⁴⁷According 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.

```

    { \exp_not:n { #1 } }
  }
}
\ExplSyntaxOff


```

Here is an example of utilisation:

```

\begin{NiceTabular}{ccc}[hvlines]
merlan & requin & cabillaud \\
baleine & \crossbox[red] & morue \\
mante & raie & poule
\end{NiceTabular}

```

merlan	requin	cabillaud
baleine		morue
mante	raie	poule

16 Technical remarks

16.1 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 job⁴⁸:

```

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

```

The heavy vertical rule won't extend in the exterior rows.⁴⁹

```

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

```

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

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.

16.2 Diagonal lines

By default, all the diagonal lines⁵⁰ 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):

```

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

```

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

⁴⁸The command `\vrule` is a TeX (and not LaTeX) command.

⁴⁹Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

⁵⁰We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in the `\CodeAfter`.

```

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

```

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

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

The same example without parallelization:

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

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]`.

16.3 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`.
- A cell of a column of type `p`, `m` or `t` is always considered as not empty. *Caution* : One should not rely upon that point because it may change in a future version of `nicematrix`.

16.4 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 idea⁵¹. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces

⁵¹In 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).*

with explicit instructions `\hskip -\arraycolsep`⁵². 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).

16.5 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=;}
```

The package `nicematrix` is not compatible with the class `ieeeaccess` (because that class is not compatible with PGF/Tikz).⁵³

In order to use `nicematrix` with the class `aastex631`, you have to add the following lines in the preamble of your document :

```
\BeforeBegin{NiceTabular}{\let\begin\BeginEnvironment\let\end\EndEnvironment}
\BeforeBegin{NiceArray}{\let\begin\BeginEnvironment}
\BeforeBegin{NiceMatrix}{\let\begin\BeginEnvironment}
```

In order to use `nicematrix` with the class `sn-jnl`, `pgf` must be loaded before the `\documentclass`:

```
\RequirePackage{pgf}
\documentclass{sn-jnl}
```

17 Examples

17.1 Utilisation of the key “tikz” of the command `\Block`

The key `tikz` of the command `\Block` is available only when Tikz is loaded.⁵⁴ For the following example, you need also the Tikz library `patterns`

```
\usetikzlibrary{patterns}

\ttfamily \small
\begin{NiceTabular}{X[m]X[m]X[m]}[hvlines, cell-space-limits=3pt]
  \Block[tikz={pattern=grid,pattern color=lightgray}]{ }
    {pattern = grid,\ \ pattern color = lightgray}
& \Block[tikz={pattern = north west lines,pattern color=blue}]{ }
    {pattern = north west lines,\ \ pattern color = blue}
& \Block[tikz={outer color = red!50, inner color=white }]{2-1}
    {outer color = red!50,\ \ inner color = white} \ \
```

⁵²And 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.

⁵³See <https://tex.stackexchange.com/questions/528975/error-loading-tikz-in-ieeeaccess-class>

⁵⁴By default, `nicematrix` only loads PGF, which is a sub-layer of Tikz.

```

\Block[tikz={pattern = sixpointed stars, pattern color = blue!15}]{ }
{pattern = sixpointed stars,\ pattern color = blue!15}
& \Block[tikz={left color = blue!50}]{ }
{left color = blue!50} \
\end{NiceTabular}

```

<pre> pattern = grid, pattern color = lightgray </pre>	<pre> pattern = north west lines, pattern color = blue </pre>	<pre> outer color = red!50, inner color = white </pre>
<pre> * pattern = sixpointed stars, * pattern color = blue!15 * </pre>	<pre> left color = blue!50 </pre>	

17.2 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 12 p. 30.

Let's consider that we wish to number the notes of a tabular with stars.⁵⁵

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 ⁵⁶

```

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

```

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

```

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

\begin{NiceTabular}{{}}{llr}}
\toprule \RowStyle{\bfseries}
Last name & First name & Birth day \
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.}
& Jacques & 5 juin 1962 \

```

⁵⁵Of course, it's realistic only when there is very few notes in the tabular.

⁵⁶In fact: the value of its argument.

```
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

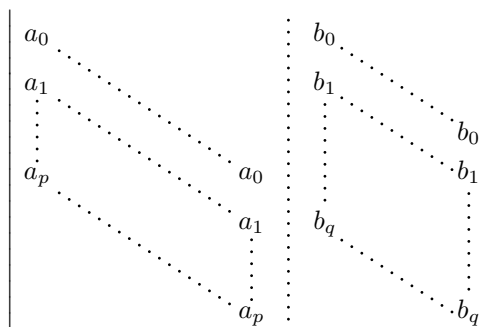
*Achard is an old family of the Poitou.

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

17.3 Dotted lines

An example with the resultant of two polynoms:

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



An example for a linear system:

```
\begin{pNiceArray}{*6c|c}[nullify-dots,last-col,code-for-last-col=\scriptstyle]
1 & & 1 & 1 & \Cdots & & 1 & & 0 & & & \\
0 & & 1 & 0 & \Cdots & & 0 & & & & & L_2 \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}
```

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

17.4 Dotted lines which are no longer dotted

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

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

$$\begin{pmatrix} 1 & & & \\ & 0 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \begin{array}{l} \leftarrow i \\ \leftarrow j \end{array}$$

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

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

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

17.5 Stacks of matrices

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

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

```

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

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

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

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

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

\end{NiceMatrixBlock}

```

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

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

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

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

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

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

```

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

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

...
\end{NiceMatrixBlock}

```

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

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

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

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

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

```

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

```

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

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

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

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

In this tabular, the instructions `\SubMatrix` are executed after the composition of the tabular and, thus, the vertical rules are drawn without adding space between the columns.

New 6.2 In fact, it's possible, with the key `vlines-in-sub-matrix`, to choice a letter in the preamble of the array to specify vertical rules which will be drawn in the `\SubMatrix` only (by adding space between the columns).

```

\setlength{\extrarowheight}{1mm}
\[\begin{NiceArray}
[
vlines-in-sub-matrix=I,
last-col,
code-for-last-col = \scriptstyle \color{blue}
]
{rrrrIr}
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 \\
12 & -8 & 7 & & 5 & 3 \\
0 & 64 & & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 & L_3 \gets 3L_2+L_3 \\
12 & -8 & 7 & & 5 & 3 \\
0 & 64 & & -41 & 1 & 19 \\
\CodeAfter
\SubMatrix({1-1}{4-5})
\SubMatrix({5-1}{8-5})
\SubMatrix({9-1}{11-5})
\SubMatrix({12-1}{13-5})
\end{NiceArray}\]

```

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

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

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

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

17.6 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 block⁵⁷).

```

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

```

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

⁵⁷We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

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 commands `\hline` and `\Hline`, the specifier “|” and the options `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` spread the cells.⁵⁸

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.⁵⁹

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}
```

We create a rectangular Tikz node which encompasses the nodes of the second row by using the tools of the Tikz library `fit`. Those nodes are not available by default in the `\CodeBefore` (for efficiency). We have to require their creation with the key `create-cell-nodes` of the keyword `\CodeBefore`.

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

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

⁵⁸For the command `\cline`, see the remark p. 8.

⁵⁹In Overleaf, the “built-in” PDF viewer does not show transparency. You can switch to the “native” viewer in that case.

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

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

```
\[ \begin{pNiceArray}{ccc}[last-col]
\CodeBefore [create-cell-nodes]
\begin{tikzpicture}
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
\Body
a & a + b & a + b + c & L_1 \\
a & a & a + b & L_2 \\
a & a & a & L_3
\end{pNiceArray} \]
```

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

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

```
\[ \begin{pNiceArray}{ccc}[last-col,create-medium-nodes]
\CodeBefore [create-cell-nodes]
\begin{tikzpicture} [name suffix = -medium]
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
\Body
a & a + b & a + b + c & L_1 \\
a & a & a + b & L_2 \\
a & a & a & L_3
\end{pNiceArray} \]
```

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

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

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

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

$$L_i \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{i1} & \cdots & a_{in} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1j} & \cdots & b_{1n} \\ \vdots & & \vdots & & \vdots \\ b_{k1} & \cdots & b_{kj} & \cdots & b_{kn} \\ \vdots & & \vdots & & \vdots \\ b_{n1} & \cdots & b_{nj} & \cdots & b_{nn} \end{pmatrix} = \begin{pmatrix} \vdots \\ \vdots \\ c_{ij} \\ \vdots \end{pmatrix}$$

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

\[\begin{NiceArray}{*{6}{c}}@{\hspace{6mm}}*{5}{c}}[nullify-dots]
\CodeBefore [create-cell-nodes]
  \SubMatrix({2-7}{6-11})
  \SubMatrix({7-2}{11-6})
  \SubMatrix({7-7}{11-11})
  \begin{tikzpicture}
    \node [highlight = (9-2) (9-6)] { } ;
    \node [highlight = (2-9) (6-9)] { } ;
  \end{tikzpicture}
\Body
  & & & & & & & \color{blue}\scriptstyle C_j \\
  & & & & & b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\
  & & & & & \Vdots & & \Vdots & & \Vdots \\
  & & & & & & & b_{kj} \\
  & & & & & & & \Vdots \\
  & & & & & b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn} \\
  & a_{11} & \Cdots & & & a_{1n} \\
  & \Vdots & & & & \Vdots \\
\color{blue}\scriptstyle L_i
  & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} & \Cdots & & c_{ij} \\
  & \Vdots & & & & \Vdots \\
  & a_{n1} & \Cdots & & & a_{nn}
\CodeAfter
\tikz \draw [gray,shorten > = 1mm, shorten < = 1mm] (9-4.north) to [bend left] (4-9.west) ;
\end{NiceArray}\]
```

18 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 the L3 programming layer.

```
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_@@_tabularx_loaded_bool
28 \bool_new:N \c_@@_tikz_loaded_bool
29 \bool_new:N \c_@@_varwidth_loaded_bool
30 \AtBeginDocument
31 {
32   \ifpackageloaded { varwidth }
33     { \bool_set_true:N \c_@@_varwidth_loaded_bool }
```

```

34     { }
35     \@ifpackageloaded { arydshln }
36     { \bool_set_true:N \c_@@_arydshln_loaded_bool }
37     { }
38     \@ifpackageloaded { booktabs }
39     { \bool_set_true:N \c_@@_booktabs_loaded_bool }
40     { }
41     \@ifpackageloaded { enumitem }
42     { \bool_set_true:N \c_@@_enumitem_loaded_bool }
43     { }
44     \@ifpackageloaded { tabularx }
45     { \bool_set_true:N \c_@@_tabularx_loaded_bool }
46     { }
47     \@ifpackageloaded { tikz }
48     {

```

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. The tokens `\exp_not:N` are mandatory.

```

49     \bool_set_true:N \c_@@_tikz_loaded_bool
50     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
51     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
52   }
53   {
54     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
55     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
56   }
57 }

```

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`).

```

58 \bool_new:N \c_@@_revtex_bool
59 \@ifclassloaded { revtex4-1 }
60 { \bool_set_true:N \c_@@_revtex_bool }
61 { }
62 \@ifclassloaded { revtex4-2 }
63 { \bool_set_true:N \c_@@_revtex_bool }
64 { }

```

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

```

65 \cs_if_exist:NT \rvtx@iffomat@geq { \bool_set_true:N \c_@@_revtex_bool }

66 \cs_generate_variant:Nn \tl_if_single_token_p:n { V }

```

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

```

67 \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.

```

68 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
69 {
70   \iow_now:Nn \@mainaux
71   {
72     \ExplSyntaxOn

```



```

73     \cs_if_free:NT \pgfsyspdfmark
74     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
75     \ExplSyntaxOff
76   }
77   \cs_gset_eq:NN \@_provide_pgfsyspdfmark: \prg_do_nothing:
78 }

```

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.

```

79 \ProvideDocumentCommand \iddots { }
80 {
81   \mathinner
82   {
83     \tex_mkern:D 1 mu
84     \box_move_up:nn { 1 pt } { \hbox:n { . } }
85     \tex_mkern:D 2 mu
86     \box_move_up:nn { 4 pt } { \hbox:n { . } }
87     \tex_mkern:D 2 mu
88     \box_move_up:nn { 7 pt }
89     { \vbox:n { \kern 7 pt \hbox:n { . } } }
90     \tex_mkern:D 1 mu
91   }
92 }

```

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.

```

93 \AtBeginDocument
94 {
95   \ifpackageloaded { booktabs }
96   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
97   { }
98 }
99 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
100 {
101   \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`).

```

102   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
103   {
104     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
105     { \@_old_pgfutil@check@rerun { ##1 } { ##2 } }
106   }
107 }

```

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

```

108 \bool_new:N \c_@@_colortbl_loaded_bool
109 \AtBeginDocument
110 {
111   \ifpackageloaded { colortbl }
112   { \bool_set_true:N \c_@@_colortbl_loaded_bool }
113   {

```

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.

```

114     \cs_set_protected:Npn \CT@arc@ { }
115     \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
116     \cs_set:Npn \CT@arc@ #1 #2

```

```

117     {
118         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
119         { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
120     }

```

Idem for \CT@drs@.

```

121     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
122     \cs_set:Npn \CT@drs #1 #2
123     {
124         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
125         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
126     }
127     \cs_set:Npn \hline
128     {
129         \noalign { \ifnum 0 = ` } \fi
130         \cs_set_eq:NN \hskip \vskip
131         \cs_set_eq:NN \vrule \hrule
132         \cs_set_eq:NN \@width \@height
133         { \CT@arc@ \vline }
134         \futurelet \reserved@a
135         \@xhline
136     }
137 }
138 }

```

We will use \AtBeginEnvironment. For version of LaTeX posterior to 2020-10-01, the command is available in the LaTeX kernel (l3hooks). For older versions, we load etoolbox.

```

139 \cs_if_exist:NF \AtBeginEnvironment { \RequirePackage { etoolbox } }

```

The command \AtBeginDocument will be used to patch {tabular} in order to come back to the original versions of \multicolumn in the {tabular} nested in the environments of nicematrix.

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.

```

140 \cs_set:Npn \@_standard_cline #1 { \@_standard_cline:w #1 \q_stop }
141 \cs_set:Npn \@_standard_cline:w #1-#2 \q_stop
142 {
143     \int_compare:nNnT \l_@_first_col_int = 0 { \omit & }
144     \int_compare:nNnT { #1 } > 1 { \multispan { \@_pred:n { #1 } } & }
145     \multispan { \int_eval:n { #2 - #1 + 1 } }
146     {
147         \CT@arc@
148         \leaders \hrule \@height \arrayrulewidth \hfill

```

The following \skip_horizontal:N \c_zero_dim is to prevent a potential \unskip to delete the \leaders⁶⁰

```

149     \skip_horizontal:N \c_zero_dim
150 }

```

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.

```

151     \everycr { }
152     \cr
153     \noalign { \skip_vertical:N -\arrayrulewidth }
154 }

```

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.

```

155 \cs_set:Npn \@_cline

```

⁶⁰See question 99041 on TeX StackExchange.

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`.

```
156 { \@@_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* or the form *i*.

```
157 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
158 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
159 {
160   \tl_if_empty:nTF { #3 }
161     { \@@_cline_iii:w #1|#2-#2 \q_stop }
162     { \@@_cline_ii:w #1|#2-#3 \q_stop }
163 }
164 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
165   { \@@_cline_iii:w #1|#2-#3 \q_stop }
166 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
167 {
```

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).

```
168   \int_compare:nNnT { #1 } < { #2 }
169     { \multispan { \int_eval:n { #2 - #1 } } & }
170   \multispan { \int_eval:n { #3 - #2 + 1 } }
171   {
172     \CT@arc@
173     \leaders \hrule \@height \arrayrulewidth \hfill
174     \skip_horizontal:N \c_zero_dim
175   }
```

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

```
176   \peek_meaning_remove_ignore_spaces:NTF \cline
177     { & \@@_cline_i:en { \@@_succ:n { #3 } } }
178     { \everycr { } \cr }
179 }
180 \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.

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

The following command is a small shortcut.

```
183 \cs_new:Npn \@@_math_toggle_token:
184   { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }
```

```
185 \cs_new_protected:Npn \@@_set_CT@arc@:
186   { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
187 \cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
188   { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
189 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
190   { \cs_set:Npn \CT@arc@ { \color { #1 } } }
```

```
191 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor
```

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`.

```
192 \bool_new:N \c_@@_siunitx_loaded_bool
193 \AtBeginDocument
194 {
195   \ifpackageloaded { siunitx }
```

```

196     { \bool_set_true:N \c_@@_siunitx_loaded_bool }
197     { }
198 }

```

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

```

199 \AtBeginDocument
200 {
201     \bool_if:nTF { ! \c_@@_siunitx_loaded_bool }
202     { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
203     {
204         \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
205         {
206             \renewcommand*{\NC@rewrite@S}[1] []
207             {
\@temptokena is a toks (not supported by the L3 programming layer).
208                 \@temptokena \exp_after:wN
209                 { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
210                 \NC@find
211             }
212         }
213     }
214 }

```

The following code is used to define `\c_@@_table_collect_begin_tl` and `\c_@@_table_print_tl` when the version of `siunitx` is prior to 3.0. The command `\@@_adapt_S_column` is used in the environment `{NiceArrayWithDelims}`.

```

215 \AtBeginDocument
216 {
217     \cs_set_eq:NN \@@_adapt_S_column: \prg_do_nothing:
218     \bool_lazy_and:nnT
219     { \c_@@_siunitx_loaded_bool }
220     { ! \cs_if_exist_p:N \siunitx_cell_begin:w }
221     {
222         \cs_set_protected:Npn \@@_adapt_S_column:
223         {
224             \group_begin:
225             \@temptokena = { }
226             \cs_set_eq:NN \NC@find \prg_do_nothing:
227             \NC@rewrite@S { }
228             \tl_gset:NV \g_tmpa_tl \@temptokena
229             \group_end:
230             \tl_new:N \c_@@_table_collect_begin_tl
231             \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
232             \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
233             \tl_new:N \c_@@_table_print_tl
234             \tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl { -1 } }
235             \cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
236         }
237     }
238 }

```

Parameters

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

```

239 \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).

```

240 \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.

```
241 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
242 { \int_use:N \g_@@_env_int }
```

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

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

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

```
245 \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).

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

The dimension `\l_@@_col_width_dim` will be available in each cell which belongs to a column of fixed width: `w{...}{...}`, `W{...}{...}`, `p{}`, `m{}`, `b{}` but also `X` (when the actual width of that column is known, that is to say after the first compilation). It's the width of that column. It will be used by some commands `\Block`. A non positive value means that the column has no fixed width (it's a column of type `c`, `r`, `l`, etc.).

```
247 \dim_new:N \l_@@_col_width_dim
248 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
249 \int_new:N \g_@@_row_total_int
250 \int_new:N \g_@@_col_total_int
```

The following counter corresponds to the key `nb-rows` of the command `\RowStyle`.

```
251 \int_new:N \l_@@_key_nb_rows_int
```

The following token list will contain the type of horizontal alignment of the current cell as provided by the corresponding column. The possible values are `r`, `l`, `c`. For exemple, a column `p[1]{3cm}` will provide the value `l` for all the cells of the column.

```
252 \str_new:N \l_@@_hpos_cell_str
253 \str_set:Nn \l_@@_hpos_cell_str { 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`.

```
254 \dim_new:N \g_@@_blocks_wd_dim
```

Idem pour the mono-row blocks.

```
255 \dim_new:N \g_@@_blocks_ht_dim
256 \dim_new:N \g_@@_blocks_dp_dim
```

The following dimension correspond to the key `width` (which may be fixed in `\NiceMatrixOptions` but also in an environment `{NiceTabular}`).

```
257 \dim_new:N \l_@@_width_dim
```

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

```
258 \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.

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

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

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

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

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

```
261 \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.

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

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

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

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

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

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag.

```
265 \bool_new:N \l_@@_X_column_bool
```

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The content of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }`).

```
266 \tl_new:N \g_@@_aux_tl
```

```
267 \cs_new_protected:Npn \@@_test_if_math_mode:
268 {
269   \if_mode_math: \else:
270     \@@_fatal:n { Outside-math-mode }
271   \fi:
272 }
```

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

```
273 \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.

```
274 \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”.

```
275 \colorlet { nicematrix-last-col } { . }
276 \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`).

```
277 \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*.

```
278 \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).

```
279 \cs_new:Npn \@@_full_name_env:
280 {
281   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
282   { command \space \c_backslash_str \g_@@_name_env_str }
283   { environment \space \{ \g_@@_name_env_str \} }
284 }
```

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`). That parameter is *public*.

```
285 \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.

```
286 \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.

```
287 \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.

```
288 \int_new:N \l_@@_old_iRow_int
289 \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.

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

The sum of the weights of all the X-columns in the preamble. The weight of a X-column is given as optional argument between square brackets. The default value, of course, is 1.

```
291 \int_new:N \g_@@_total_X_weight_int
```

If there is at least one X-column in the preamble of the array, the following flag will be raised via the `aux` file. The length `\l_@@_x_columns_dim` will be the width of X-columns of weight 1 (the width of a column of weight n will be that dimension multiplied by n). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
292 \bool_new:N \l_@@_X_columns_aux_bool
293 \dim_new:N \l_@@_X_columns_dim
```

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.

```
294 \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).

```
295 \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).

```
296 \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`).

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

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

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

```
299 \tl_new:N \g_@@_row_style_tl
```

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

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

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

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

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

The L3 programming layer provides scratch dimensions `\l_tmpa_dim` and `\l_tmpb_dim`. We creates two more in the same spirit (if they don't exist yet: that's why we use `\dim_zero_new:N`).

```
304 \dim_zero_new:N \l_tmpc_dim
```

```
305 \dim_zero_new:N \l_tmpd_dim
```

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

```
306 \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”.

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

```
308 \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.

```
309 \seq_new:N \g_@@_blocks_seq
```


We also manage a sequence of the *positions* of the blocks. In that sequence, each block is represented by only five components: $\{imin\}\{jmin\}\{imax\}\{jmax\}\{name\}$. A block with the key `hvlines` won't appear in that sequence (otherwise, the lines in that block would not be drawn!).

```
310 \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 five components: $\{imin\}\{jmin\}\{imax\}\{jmax\}\{name\}$.

```
311 \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.

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

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

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

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

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

The following flag will be raised if the key `width` is used in an environment `\NiceTabular` (not in a command `\NiceMatrixOptions`). You use it to raise an error when this key is used while no column `X` is used.

```
315 \bool_new:N \l_@@_width_used_bool
```

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).

```
316 \seq_new:N \g_@@_multicolumn_cells_seq
```

```
317 \seq_new:N \g_@@_multicolumn_sizes_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`).

```
318 \int_new:N \l_@@_row_min_int
```

```
319 \int_new:N \l_@@_row_max_int
```

```
320 \int_new:N \l_@@_col_min_int
```

```
321 \int_new:N \l_@@_col_max_int
```

The following sequence will be used when the command `\SubMatrix` is used in the `\CodeBefore` (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.

```
322 \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).

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

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

```
324 \tl_new:N \l_@@_fill_tl
325 \tl_new:N \l_@@_draw_tl
326 \seq_new:N \l_@@_tikz_seq
327 \clist_new:N \l_@@_borders_clist
328 \dim_new:N \l_@@_rounded_corners_dim
```

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

The following token list correspond to the key `color` of the command `\Block`.

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

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

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

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

```
331 \str_new:N \l_@@_hpos_block_str
332 \str_set:Nn \l_@@_hpos_block_str { c }
333 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

For the vertical position, the possible values are `c`, `t` and `b`. Of course, it would be interesting to program a key `T` and a key `B`.

```
334 \tl_new:N \l_@@_vpos_of_block_tl
335 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

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

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

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

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

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

339 \dim_new:N \l_@@_submatrix_extra_height_dim
340 \dim_new:N \l_@@_submatrix_left_xshift_dim
341 \dim_new:N \l_@@_submatrix_right_xshift_dim
342 \clist_new:N \l_@@_hlines_clist
343 \clist_new:N \l_@@_vlines_clist
344 \clist_new:N \l_@@_submatrix_hlines_clist
345 \clist_new:N \l_@@_submatrix_vlines_clist
```

The following flag will be used by (for instance) `\@@_vline_ii:nnnn`. When `\l_@@_dotted_bool` is `true`, a dotted line (with our system) will be drawn.

```
346 \bool_new:N \l_@@_dotted_bool
```

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.

```
347 \int_new:N \l_@@_first_row_int
348 \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.

```
349 \int_new:N \l_@@_first_col_int
350 \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).

```
351 \int_new:N \l_@@_last_row_int
352 \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”.⁶¹

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

Idem for `\l_@@_last_col_without_value_bool`

```
354 \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.

```
355 \int_new:N \l_@@_last_col_int
356 \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}
```

⁶¹We 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.

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.

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

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

Some utilities

```
358 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
359 {
360   \tl_set:Nn \l_tmpa_tl { #1 }
361   \tl_set:Nn \l_tmpb_tl { #2 }
362 }
```

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers.

```
363 \cs_new_protected:Npn \@@_expand_clist:N #1
364 {
365   \clist_if_in:NnF #1 { all }
366   {
367     \clist_clear:N \l_tmpa_clist
368     \clist_map_inline:Nn #1
369     {
370       \tl_if_in:nnTF { ##1 } { - }
371       { \@@_cut_on_hyphen:w ##1 \q_stop }
372       {
373         \tl_set:Nn \l_tmpa_tl { ##1 }
374         \tl_set:Nn \l_tmpb_tl { ##1 }
375       }
376       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
377       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
378     }
379     \tl_set_eq:NN #1 \l_tmpa_clist
380   }
381 }
```

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.

```
382 \newcounter { tablarnote }
```

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

```
383 \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.

```
384 \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).

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

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

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

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

```
387 \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`.

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

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

```
389 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }
```

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

```
390 \AtBeginDocument
391 {
392   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
393   {
394     \NewDocumentCommand \tabularnote { m }
395     { \@@_error:n { enumitem-not-loaded } }
396   }
397   {
```

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

```
398   \newlist { tabularnotes } { enumerate } { 1 }
399   \setlist [ tabularnotes ]
400   {
401     topsep = 0pt ,
402     noitemsep ,
403     leftmargin = * ,
404     align = left ,
405     labelsep = 0pt ,
406     label =
407     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
408   }
409   \newlist { tabularnotes* } { enumerate* } { 1 }
410   \setlist [ tabularnotes* ]
411   {
412     afterlabel = \nobreak ,
413     itemjoin = \quad ,
414     label =
415     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
416   }
```

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.⁶²

⁶²We should try to find a solution to that problem.

```

417 \NewDocumentCommand \tabularnote { m }
418 {
419     \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
420     { \@@_error:n { tabularnote~forbidden } }
421     {

```

`\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).

```

422         \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`.

```

423         \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
424         \peek_meaning:NF \tabularnote
425         {

```

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.

```

426             \hbox_set:Nn \l_tmpa_box
427             {

```

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

```

428                 \@@_notes_label_in_tabular:n
429                 {
430                     \stepcounter { tabularnote }
431                     \@@_notes_style:n { tabularnote }
432                     \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
433                     {
434                         ,
435                         \stepcounter { tabularnote }
436                         \@@_notes_style:n { tabularnote }
437                     }
438                 }
439             }

```

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.

```

440                 \addtocounter { tabularnote } { -1 }
441                 \refstepcounter { tabularnote }
442                 \int_zero:N \l_@@_number_of_notes_int
443                 \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).

```

444                 \skip_horizontal:n { \box_wd:N \l_tmpa_box }
445             }
446         }
447     }
448 }
449 }

```

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.

```

450 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
451 {

```

```

452 \begin { pgfscope }
453 \pgfset
454 {
455     outer-sep = \c_zero_dim ,
456     inner-sep = \c_zero_dim ,
457     minimum-size = \c_zero_dim
458 }
459 \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
460 \pgfnode
461 { rectangle }
462 { center }
463 {
464     \vbox_to_ht:nn
465     { \dim_abs:n { #5 - #3 } }
466     {
467         \vfill
468         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
469     }
470 }
471 { #1 }
472 { }
473 \end { pgfscope }
474 }

```

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.

```

475 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
476 {
477     \begin { pgfscope }
478     \pgfset
479     {
480         outer-sep = \c_zero_dim ,
481         inner-sep = \c_zero_dim ,
482         minimum-size = \c_zero_dim
483     }
484     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
485     \pgfpointdiff { #3 } { #2 }
486     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
487     \pgfnode
488     { rectangle }
489     { center }
490     {
491         \vbox_to_ht:nn
492         { \dim_abs:n \l_tmpb_dim }
493         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
494     }
495     { #1 }
496     { }
497     \end { pgfscope }
498 }

```

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.

```

499 \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`.

```
500 \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`).

```
501 \dim_new:N \l_@@_cell_space_top_limit_dim
502 \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.

```
503 \dim_new:N \l_@@_inter_dots_dim
504 \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).

```
505 \dim_new:N \l_@@_xdots_shorten_dim
506 \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.

```
507 \dim_new:N \l_@@_radius_dim
508 \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.

```
509 \tl_new:N \l_@@_xdots_line_style_tl
510 \tl_const:Nn \c_@@_standard_tl { standard }
511 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
512 \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).

```
513 \tl_new:N \l_@@_baseline_tl
514 \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`).

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

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

```
516 \bool_new:N \l_@@_parallelize_diags_bool
517 \bool_set_true:N \l_@@_parallelize_diags_bool
```


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

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

```
519 \dim_new:N \l_@@_notes_above_space_dim
```

```
520 \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.).

```
521 \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).

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

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

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

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

```
524 \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”.

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

```
526 \bool_new:N \l_@@_large_nodes_bool
```

The boolean `\l_@@_except_borders_bool` will be raised when the key `hvlines-except-borders` will be used (but that key has also other effects).

```
527 \bool_new:N \l_@@_except_borders_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`).

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

```
529 \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`.

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

```
531 \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.

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

```
533 \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 “:”.

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

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

```
535 \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 function 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.

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

```
537 \keys_define:nn { NiceMatrix / xdots }
538 {
539   line-style .code:n =
540   {
541     \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.

```
542   { \cs_if_exist_p:N \tikzpicture }
543   { \str_if_eq_p:nn { #1 } { standard } }
544   { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
545   { \@@_error:n { bad-option-for-line-style } }
546   } ,
547   line-style .value_required:n = true ,
548   color .tl_set:N = \l_@@_xdots_color_tl ,
549   color .value_required:n = true ,
550   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
551   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 `_`.

```
552   down .tl_set:N = \l_@@_xdots_down_tl ,
553   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).

```
554   draw-first .code:n = \prg_do_nothing: ,
555   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
556 }
```

```
557 \keys_define:nn { NiceMatrix / rules }
558 {
559   color .tl_set:N = \l_@@_rules_color_tl ,
560   color .value_required:n = true ,
561   width .dim_set:N = \arrayrulewidth ,
562   width .value_required:n = true
563 }
```

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

```
564 \keys_define:nn { NiceMatrix / Global }
565 {
566   delimiters .code:n =
567   \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
568   delimiters .value_required:n = true ,
569   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
570   rules .value_required:n = true ,
```

```

571 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
572 standard-cline .default:n = true ,
573 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
574 cell-space-top-limit .value_required:n = true ,
575 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
576 cell-space-bottom-limit .value_required:n = true ,
577 cell-space-limits .meta:n =
578 {
579     cell-space-top-limit = #1 ,
580     cell-space-bottom-limit = #1 ,
581 } ,
582 cell-space-limits .value_required:n = true ,
583 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
584 light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
585 light-syntax .default:n = true ,
586 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
587 end-of-row .value_required:n = true ,
588 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
589 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
590 last-row .int_set:N = \l_@@_last_row_int ,
591 last-row .default:n = -1 ,
592 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
593 code-for-first-col .value_required:n = true ,
594 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
595 code-for-last-col .value_required:n = true ,
596 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
597 code-for-first-row .value_required:n = true ,
598 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
599 code-for-last-row .value_required:n = true ,
600 hlines .clist_set:N = \l_@@_hlines_clist ,
601 vlines .clist_set:N = \l_@@_vlines_clist ,
602 hlines .default:n = all ,
603 vlines .default:n = all ,
604 vlines-in-sub-matrix .code:n =
605 {
606     \tl_if_single_token:nTF { #1 }
607     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
608     { \@@_error:n { One~letter~allowed } }
609 } ,
610 vlines-in-sub-matrix .value_required:n = true ,
611 hvlines .code:n =
612 {
613     \clist_set:Nn \l_@@_vlines_clist { all }
614     \clist_set:Nn \l_@@_hlines_clist { all }
615 } ,
616 hvlines-except-borders .code:n =
617 {
618     \clist_set:Nn \l_@@_vlines_clist { all }
619     \clist_set:Nn \l_@@_hlines_clist { all }
620     \bool_set_true:N \l_@@_except_borders_bool
621 } ,
622 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.

```

623 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
624 renew-dots .value_forbidden:n = true ,
625 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
626 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
627 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
628 create-extra-nodes .meta:n =
629 { create-medium-nodes , create-large-nodes } ,
630 left-margin .dim_set:N = \l_@@_left_margin_dim ,

```

```

631 left-margin .default:n = \arraycolsep ,
632 right-margin .dim_set:N = \l_@@_right_margin_dim ,
633 right-margin .default:n = \arraycolsep ,
634 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
635 margin .default:n = \arraycolsep ,
636 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
637 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
638 extra-margin .meta:n =
639   { extra-left-margin = #1 , extra-right-margin = #1 } ,
640 extra-margin .value_required:n = true ,
641 }

```

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

```

642 \keys_define:nn { NiceMatrix / Env }
643   {

```

The key `hvlines-except-corners` is now deprecated (use `hvlines` and `corners` instead).

```

644   hvlines-except-corners .code:n =
645     {
646       \clist_set:Nn \l_@@_corners_clist { #1 }
647       \clist_set:Nn \l_@@_vlines_clist { all }
648       \clist_set:Nn \l_@@_hlines_clist { all }
649     } ,
650   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
651   corners .clist_set:N = \l_@@_corners_clist ,
652   corners .default:n = { NW , SW , NE , SE } ,
653   code-before .code:n =
654     {
655       \tl_if_empty:nF { #1 }
656       {
657         \tl_put_right:Nn \l_@@_code_before_tl { #1 }
658         \bool_set_true:N \l_@@_code_before_bool
659       }
660     } ,

```

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

```

661   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
662   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
663   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
664   baseline .tl_set:N = \l_@@_baseline_tl ,
665   baseline .value_required:n = true ,
666   columns-width .code:n =
667     \tl_if_eq:nnTF { #1 } { auto }
668     { \bool_set_true:N \l_@@_auto_columns_width_bool }
669     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
670   columns-width .value_required:n = true ,
671   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.

```

672   \legacy_if:nF { measuring@ }
673   {
674     \str_set:Nn \l_tmpa_str { #1 }
675     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
676     { \@@_error:nn { Duplicate-name } { #1 } }
677     { \seq_gput_left:N \g_@@_names_seq \l_tmpa_str }
678     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
679   } ,
680   name .value_required:n = true ,
681   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
682   code-after .value_required:n = true ,

```

```

683 colortbl-like .code:n =
684   \bool_set_true:N \l_@@_colortbl_like_bool
685   \bool_set_true:N \l_@@_code_before_bool ,
686   colortbl-like .value_forbidden:n = true
687 }
688 \keys_define:nn { NiceMatrix / notes }
689 {
690   para .bool_set:N = \l_@@_notes_para_bool ,
691   para .default:n = true ,
692   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
693   code-before .value_required:n = true ,
694   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
695   code-after .value_required:n = true ,
696   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
697   bottomrule .default:n = true ,
698   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
699   style .value_required:n = true ,
700   label-in-tabular .code:n =
701     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
702   label-in-tabular .value_required:n = true ,
703   label-in-list .code:n =
704     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
705   label-in-list .value_required:n = true ,
706   enumitem-keys .code:n =
707     {
708       \bool_if:NTF \c_@@_in_preamble_bool
709       {
710         \AtBeginDocument
711         {
712           \bool_if:NT \c_@@_enumitem_loaded_bool
713           { \setlist* [ tabularnotes ] { #1 } }
714         }
715       }
716       {
717         \bool_if:NT \c_@@_enumitem_loaded_bool
718         { \setlist* [ tabularnotes ] { #1 } }
719       }
720     } ,
721   enumitem-keys .value_required:n = true ,
722   enumitem-keys-para .code:n =
723     {
724       \bool_if:NTF \c_@@_in_preamble_bool
725       {
726         \AtBeginDocument
727         {
728           \bool_if:NT \c_@@_enumitem_loaded_bool
729           { \setlist* [ tabularnotes* ] { #1 } }
730         }
731       }
732       {
733         \bool_if:NT \c_@@_enumitem_loaded_bool
734         { \setlist* [ tabularnotes* ] { #1 } }
735       }
736     } ,
737   enumitem-keys-para .value_required:n = true ,
738   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
739 }
740 \keys_define:nn { NiceMatrix / delimiters }
741 {
742   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
743   max-width .default:n = true ,
744   color .tl_set:N = \l_@@_delimiters_color_tl ,
745   color .value_required:n = true ,

```

746 }

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

```

747 \keys_define:nn { NiceMatrix }
748 {
749   NiceMatrixOptions .inherit:n =
750     { NiceMatrix / Global } ,
751   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
752   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
753   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
754   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
755   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
756   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
757   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
758   NiceMatrix .inherit:n =
759     {
760       NiceMatrix / Global ,
761       NiceMatrix / Env ,
762     } ,
763   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
764   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
765   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
766   NiceTabular .inherit:n =
767     {
768       NiceMatrix / Global ,
769       NiceMatrix / Env
770     } ,
771   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
772   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
773   NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
774   NiceArray .inherit:n =
775     {
776       NiceMatrix / Global ,
777       NiceMatrix / Env ,
778     } ,
779   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
780   NiceArray / rules .inherit:n = NiceMatrix / rules ,
781   NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
782   pNiceArray .inherit:n =
783     {
784       NiceMatrix / Global ,
785       NiceMatrix / Env ,
786     } ,
787   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
788   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
789   pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
790 }

```

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

```

791 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
792 {
793   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
794   width .value_required:n = true ,
795   last-col .code:n = \tl_if_empty:nF { #1 }
796     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
797     \int_zero:N \l_@@_last_col_int ,
798   small .bool_set:N = \l_@@_small_bool ,
799   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.

```

800   renew-matrix .code:n = \@@_renew_matrix: ,
801   renew-matrix .value_forbidden:n = true ,

```

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

```

802   transparent .code:n =
803   {
804     \@@_renew_matrix:
805     \bool_set_true:N \l_@@_renew_dots_bool
806     \@@_error:n { Key-transparent }
807   } ,
808   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}`.

```

809   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.

```

810   columns-width .code:n =
811   \tl_if_eq:nnTF { #1 } { auto }
812   { \@@_error:n { Option-auto-for-columns-width } }
813   { \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.

```

814   allow-duplicate-names .code:n =
815   \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
816   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`).

```

817   letter-for-dotted-lines .code:n =
818   {
819     \tl_if_single_token:nTF { #1 }
820     { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
821     { \@@_error:n { One-letter-allowed } }
822   } ,
823   letter-for-dotted-lines .value_required:n = true ,
824   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
825   notes .value_required:n = true ,
826   sub-matrix .code:n =
827   \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
828   sub-matrix .value_required:n = true ,
829   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
830 }

831 \str_new:N \l_@@_letter_for_dotted_lines_str
832 \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.

```

833 \NewDocumentCommand \NiceMatrixOptions { m }
834 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

835 \keys_define:nn { NiceMatrix / NiceMatrix }
836 {
837   last-col .code:n = \tl_if_empty:nTF {#1}
838     {
839       \bool_set_true:N \l_@@_last_col_without_value_bool
840       \int_set:Nn \l_@@_last_col_int { -1 }
841     }
842     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
843   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
844   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
845   small .bool_set:N = \l_@@_small_bool ,
846   small .value_forbidden:n = true ,
847   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
848 }

```

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

```

849 \keys_define:nn { NiceMatrix / NiceArray }
850 {

```

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.

```

851   small .bool_set:N = \l_@@_small_bool ,
852   small .value_forbidden:n = true ,
853   last-col .code:n = \tl_if_empty:nF { #1 }
854     { \@@_error:n { last-col~non-empty~for~NiceArray } }
855     { \int_zero:N \l_@@_last_col_int ,
856   notes / para .bool_set:N = \l_@@_notes_para_bool ,
857   notes / para .default:n = true ,
858   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
859   notes / bottomrule .default:n = true ,
860   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
861   tabularnote .value_required:n = true ,
862   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
863   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
864   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
865 }

866 \keys_define:nn { NiceMatrix / pNiceArray }
867 {
868   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
869   last-col .code:n = \tl_if_empty:nF {#1}
870     { \@@_error:n { last-col~non-empty~for~NiceArray } }
871     { \int_zero:N \l_@@_last_col_int ,
872   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
873   small .bool_set:N = \l_@@_small_bool ,
874   small .value_forbidden:n = true ,
875   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
876   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
877   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
878 }

```

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

```

879 \keys_define:nn { NiceMatrix / NiceTabular }
880 {

```

The dimension width will be used if at least a column of type X is used. If there is no column of type X, an error will be raised.

```

881   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }

```



```

882             \bool_set_true:N \l_@@_width_used_bool ,
883     width .value_required:n = true ,
884     notes / para .bool_set:N = \l_@@_notes_para_bool ,
885     notes / para .default:n = true ,
886     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
887     notes / bottomrule .default:n = true ,
888     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
889     tabularnote .value_required:n = true ,
890     last-col .code:n = \tl_if_empty:nF {#1}
891                     { \@@_error:n { last-col~non~empty~for~NiceArray } }
892                     \int_zero:N \l_@@_last_col_int ,
893     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
894     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
895     unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
896 }

```

Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_cell_begin:w-\@@_cell_end:` 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}`).

```

897 \cs_new_protected:Npn \@@_cell_begin:w
898 {

```

The token list `\g_@@_post_action_cell_tl` will be set during the composition of the box `\l_@@_cell_box` and will be used *after* the composition in order to modify that box (that's why it's called a *post-action*).

```

899     \tl_gclear:N \g_@@_post_action_cell_tl

```

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

```

900     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

901     \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.

```

902     \int_compare:nNnT \c@jCol = 1
903     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }

```

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the `\@@_cell_end:` (and the potential `\c_math_toggle_token` also).

```

904     \hbox_set:Nw \l_@@_cell_box
905     \bool_if:NF \l_@@_NiceTabular_bool
906     {
907         \c_math_toggle_token
908         \bool_if:NT \l_@@_small_bool \scriptstyle
909     }

```

For unexplained reason, with XeTeX (and not with the other engines), the environments of `nicematrix` were all composed in black and do not take into account the color of the encompassing text. As a workaround, you peek the color in force at the beginning of the environment and we use it now (in each cell of the array).

```

910     \color { nicematrix }
911     \g_@@_row_style_tl

```

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.

```

912   \int_compare:nNnTF \c@iRow = 0
913   {
914     \int_compare:nNnT \c@jCol > 0
915     {
916       \l_@@_code_for_first_row_tl
917       \xglobal \colorlet { nicematrix-first-row } { . }
918     }
919   }
920   {
921     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
922     {
923       \l_@@_code_for_last_row_tl
924       \xglobal \colorlet { nicematrix-last-row } { . }
925     }
926   }
927 }

```

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.

```

928 \cs_new_protected:Npn \@@_begin_of_row:
929 {
930   \int_gincr:N \c@iRow
931   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
932   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
933   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
934   \pgfpicture
935   \pgfrememberpicturepositiononpagetrue
936   \pgfcoordinate
937   { \@@_env: - row - \int_use:N \c@iRow - base }
938   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
939   \str_if_empty:NF \l_@@_name_str
940   {
941     \pgfnodealias
942     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
943     { \@@_env: - row - \int_use:N \c@iRow - base }
944   }
945   \endpgfpicture
946 }

```

Remark: If the key `recreate-cell-nodes` of the `\CodeBefore` is used, then we will add some lines to that command.

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.

```

947 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
948 {
949   \int_compare:nNnTF \c@iRow = 0
950   {
951     \dim_gset:Nn \g_@@_dp_row_zero_dim
952     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
953     \dim_gset:Nn \g_@@_ht_row_zero_dim
954     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
955   }
956   {
957     \int_compare:nNnT \c@iRow = 1
958     {
959       \dim_gset:Nn \g_@@_ht_row_one_dim

```

```

960         { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
961     }
962 }
963 }
964 \cs_new_protected:Npn \@@_rotate_cell_box:
965 {
966     \box_rotate:Nn \l_@@_cell_box { 90 }
967     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
968     {
969         \vbox_set_top:Nn \l_@@_cell_box
970         {
971             \vbox_to_zero:n { }
972             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
973             \box_use:N \l_@@_cell_box
974         }
975     }
976     \bool_gset_false:N \g_@@_rotate_bool
977 }
978 \cs_new_protected:Npn \@@_adjust_size_box:
979 {
980     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
981     {
982         \box_set_wd:Nn \l_@@_cell_box
983         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
984         \dim_gzero:N \g_@@_blocks_wd_dim
985     }
986     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
987     {
988         \box_set_dp:Nn \l_@@_cell_box
989         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
990         \dim_gzero:N \g_@@_blocks_dp_dim
991     }
992     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
993     {
994         \box_set_ht:Nn \l_@@_cell_box
995         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
996         \dim_gzero:N \g_@@_blocks_ht_dim
997     }
998 }
999 \cs_new_protected:Npn \@@_cell_end:
1000 {
1001     \@@_math_toggle_token:
1002     \hbox_set_end:

```

The token list `\g_@@_post_action_cell_tl` is (potentially) set during the composition of the box `\l_@@_cell_box` and is used now *after* the composition in order to modify that box.

```

1003     \g_@@_post_action_cell_tl
1004     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1005     \@@_adjust_size_box:
1006     \box_set_ht:Nn \l_@@_cell_box
1007     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1008     \box_set_dp:Nn \l_@@_cell_box
1009     { \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”).

```

1010     \dim_gset:Nn \g_@@_max_cell_width_dim
1011     { \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”.

```

1012     \@@_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. Up to 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.

```

1013 \bool_if:NTF \g_@@_empty_cell_bool
1014 { \box_use_drop:N \l_@@_cell_box }
1015 {
1016     \bool_lazy_or:nnTF
1017     \g_@@_not_empty_cell_bool
1018     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1019     \@@_node_for_cell:
1020     { \box_use_drop:N \l_@@_cell_box }
1021 }
1022 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1023 \bool_gset_false:N \g_@@_empty_cell_bool
1024 \bool_gset_false:N \g_@@_not_empty_cell_bool
1025 }
```

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

```

1026 \cs_new_protected:Npn \@@_node_for_cell:
1027 {
1028     \pgfpicture
1029     \pgfsetbaseline \c_zero_dim
1030     \pgfrememberpicturepositiononpagetrue
1031     \pgfset
1032     {
1033         inner~sep = \c_zero_dim ,
1034         minimum~width = \c_zero_dim
1035     }
1036     \pgfnode
1037     { rectangle }
1038     { base }
1039     { \box_use_drop:N \l_@@_cell_box }
1040     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1041     { }
1042     \str_if_empty:NF \l_@@_name_str
1043     {
1044         \pgfnodealias
1045         { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1046         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1047     }
1048     \endpgfpicture
1049 }
```

As its name says, the following command is a patch for the command `\@@_node_for_cell:`. This patch will be appended on the left of `\@@_node_for_the_cell:` when the construction of the cell nodes (of the form `(i-j)`) in the `\CodeBefore` is required.

```

1050 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
```

```

1051 {
1052   \cs_new_protected:Npn \@@_patch_node_for_cell:
1053   {
1054     \hbox_set:Nn \l_@@_cell_box
1055     {
1056       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1057       \hbox_overlap_left:n
1058       {
1059         \pgfsys@markposition
1060         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

I don't know why the following adjustment is needed when the compilation is done with XeLaTeX or with the classical way latex, divps, ps2pdf (or Adobe Distiller). However, it seems to work.

```

1061       #1
1062     }
1063     \box_use:N \l_@@_cell_box
1064     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1065     \hbox_overlap_left:n
1066     {
1067       \pgfsys@markposition
1068       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1069       #1
1070     }
1071   }
1072 }
1073 }

```

We have no explanation for the different behaviour between the TeX engines...

```

1074 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1075 {
1076   \@@_patch_node_for_cell:n
1077   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1078 }
1079 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

1080 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1081 {
1082   \bool_if:nTF { #1 } { \tl_gput_left:cx \tl_gput_right:cx
1083     { \g_@@_#2 _ lines _ tl }
1084     {
1085       \use:c { @@ _ draw _ #2 : nnn }
1086       { \int_use:N \c@iRow }
1087       { \int_use:N \c@jCol }
1088       { \exp_not:n { #3 } }

```

```

1089     }
1090 }
1091 \cs_new_protected:Npn \@@_array:
1092 {
1093   \bool_if:NTF \l_@@_NiceTabular_bool
1094     { \dim_set_eq:NN \col@sep \tabcolsep }
1095     { \dim_set_eq:NN \col@sep \arraycolsep }
1096   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1097     { \cs_set_nopar:Npn \@halignto { } }
1098     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1099   \@tabarray
\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.
1100   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1101 }

```

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.

```

1102 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1103 \cs_new_protected:Npn \@@_create_row_node:
1104 {

```

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

```

1105   \hbox
1106   {
1107     \bool_if:NT \l_@@_code_before_bool
1108     {
1109       \vtop
1110       {
1111         \skip_vertical:N 0.5\arrayrulewidth
1112         \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
1113         \skip_vertical:N -0.5\arrayrulewidth
1114       }
1115     }
1116     \pgfpicture
1117     \pgfrememberpicturerepositiononpagetrue
1118     \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
1119     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1120     \str_if_empty:NF \l_@@_name_str
1121     {
1122       \pgfnodealias
1123       { \l_@@_name_str - row - \@@_succ:n \c@iRow }
1124       { \@@_env: - row - \@@_succ:n \c@iRow }
1125     }
1126     \endpgfpicture
1127   }
1128 }

```

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

```

1129 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1130 \cs_new_protected:Npn \@@_everycr_i:
1131 {
1132   \int_gzero:N \c@jCol
1133   \bool_gset_false:N \g_@@_after_col_zero_bool
1134   \bool_if:NF \g_@@_row_of_col_done_bool
1135   {
1136     \@@_create_row_node:

```

We don't draw now the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for theses rules (the rules will be drawn by PGF).

```

1137     \tl_if_empty:NF \l_@@_hlines_clist
1138     {
1139         \tl_if_eq:NnF \l_@@_hlines_clist { all }
1140         {
1141             \exp_args:NNx
1142             \clist_if_in:NnT
1143             \l_@@_hlines_clist
1144             { \@@_succ:n \c@iRow }
1145         }
1146     }

```

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.

```

1147     \int_compare:nNnT \c@iRow > { -1 }
1148     {
1149         \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@`.

```

1150         { \hrule height \arrayrulewidth width \c_zero_dim }
1151     }
1152 }
1153 }
1154 }
1155 }

```

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`).

```

1156 \cs_set_protected:Npn \@@_newcolumntype #1
1157 {
1158     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1159     \peek_meaning:NTF [
1160         { \newcol@ #1 }
1161         { \newcol@ #1 [ 0 ] }
1162     }

```

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

```

1163 \cs_set_protected:Npn \@@_renew_dots:
1164 {
1165     \cs_set_eq:NN \ldots \@@_Ldots
1166     \cs_set_eq:NN \cdots \@@_Cdots
1167     \cs_set_eq:NN \vdots \@@_Vdots
1168     \cs_set_eq:NN \ddots \@@_Ddots
1169     \cs_set_eq:NN \iddots \@@_Iddots
1170     \cs_set_eq:NN \dots \@@_Ldots
1171     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1172 }

```

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

```

1173 \cs_new_protected:Npn \@@_colortbl_like:
1174 {
1175     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1176     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1177     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1178 }

```

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

```

1179 \cs_new_protected:Npn \@@_pre_array_ii:
1180 {
1181 % \end{macrocode}
1182 % For unexplained reason, with XeTeX (and not with the other engines), the
1183 % environments of \pkg{nicematrix} were all composed in black and do not take
1184 % into account the color of the encompassing text. As a workaround, you peek the
1185 % color in force at the beginning of the environment and we will it in each cell.
1186 % \begin{macrocode}
1187 \xglobal \colorlet { nicematrix } { . }

```

The number of letters X in the preamble of the array.

```

1188 \int_gzero:N \g_@@_total_X_weight_int
1189 \@@_expand_clist:N \l_@@_hlines_clist
1190 \@@_expand_clist:N \l_@@_vlines_clist

```

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 ⁶³.

```

1191 \bool_if:NT \c_@@_booktabs_loaded_bool
1192 { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1193 \box_clear_new:N \l_@@_cell_box
1194 \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}`).

```

1195 \bool_if:NT \l_@@_small_bool
1196 {
1197     \cs_set_nopar:Npn \arraystretch { 0.47 }
1198     \dim_set:Nn \arraycolsep { 1.45 pt }
1199 }

1200 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1201 {
1202     \tl_put_right:Nn \@@_begin_of_row:
1203     {
1204         \pgfsys@markposition
1205         { \@@_env: - row - \int_use:N \c@iRow - base }
1206     }
1207 }

```

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`.

```

1208 \cs_set_nopar:Npn \ialign
1209 {
1210     \bool_if:NTF \c_@@_colortbl_loaded_bool
1211     {
1212         \CT@everycr
1213         {
1214             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }

```

⁶³cf. `\nicematrix@redefine@check@rerun`


```

1215         \@@_everycr:
1216     }
1217 }
1218 { \everycr { \@@_everycr: } }
1219 \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`⁶⁴ 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`.

```

1220     \dim_gzero_new:N \g_@@_dp_row_zero_dim
1221     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1222     \dim_gzero_new:N \g_@@_ht_row_zero_dim
1223     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1224     \dim_gzero_new:N \g_@@_ht_row_one_dim
1225     \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1226     \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1227     \dim_gzero_new:N \g_@@_ht_last_row_dim
1228     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1229     \dim_gzero_new:N \g_@@_dp_last_row_dim
1230     \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`.

```

1231     \cs_set_eq:NN \ialign \@@_old_ialign:
1232     \halign
1233 }

```

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).

```

1234     \cs_set_eq:NN \@@_old_ldots \ldots
1235     \cs_set_eq:NN \@@_old_cdots \cdots
1236     \cs_set_eq:NN \@@_old_vdots \vdots
1237     \cs_set_eq:NN \@@_old_ddots \ddots
1238     \cs_set_eq:NN \@@_old_iddots \iddots
1239     \bool_if:NTF \l_@@_standard_cline_bool
1240     { \cs_set_eq:NN \cline \@@_standard_cline }
1241     { \cs_set_eq:NN \cline \@@_cline }
1242     \cs_set_eq:NN \Ldots \@@_Ldots
1243     \cs_set_eq:NN \Cdots \@@_Cdots
1244     \cs_set_eq:NN \Vdots \@@_Vdots
1245     \cs_set_eq:NN \Ddots \@@_Ddots
1246     \cs_set_eq:NN \Iddots \@@_Iddots
1247     \cs_set_eq:NN \hdottedline \@@_hdottedline:
1248     \cs_set_eq:NN \Hline \@@_Hline:
1249     \cs_set_eq:NN \Hspace \@@_Hspace:
1250     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1251     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1252     \cs_set_eq:NN \Block \@@_Block:
1253     \cs_set_eq:NN \rotate \@@_rotate:
1254     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1255     \cs_set_eq:NN \dotfill \@@_old_dotfill:
1256     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1257     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1258     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1259     \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1260     \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:

```

⁶⁴The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```
1261 \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:
```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition. The command `\AtBeginEnvironment` is the command of `l3hooks` and, if this command is not available (versions of LaTeX prior to 2020-10-01), `etoolbox` is loaded and the command `\AtBeginDocument` of `etoolbox` is used.

```
1262 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1263 \AtBeginEnvironment { tabular }
1264 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
```

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).

```
1265 \seq_gclear:N \g_@@_multicolumn_cells_seq
1266 \seq_gclear: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).

```
1267 \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`).

```
1268 \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_begin:w` executed at the beginning of each cell.

```
1269 \int_gzero_new:N \g_@@_col_total_int
1270 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1271 \@@_renew_NC@rewrite@S:
1272 \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.

```
1273 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1274 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1275 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1276 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1277 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1278 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl
```

```
1279 \tl_gclear_new:N \g_nicematrix_code_before_tl
1280 }
```

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

The command `\@@_pre_array:` will be executed after analyse of the keys of the environment.

```
1281 \cs_new_protected:Npn \@@_pre_array:
1282 {
1283   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1284   \int_gzero_new:N \c@iRow
1285   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1286   \int_gzero_new:N \c@jCol
```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_int` are *not* the numbers of the last row and last column of the array. There are only the values of the keys `last-row` and `last-column` (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of `nicematrix`. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the `aux` file (of course, it's possible only after the first compilation).

```

1287 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1288 {
1289   \bool_set_true:N \l_@@_last_row_without_value_bool
1290   \bool_if:NT \g_@@_aux_found_bool
1291     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \c_@@_size_seq 3 } }
1292 }
1293 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1294 {
1295   \bool_if:NT \g_@@_aux_found_bool
1296     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \c_@@_size_seq 6 } }
1297 }

```

If there is a exterior row, we patch a command used in `\@@_cell_begin:w` in order to keep track of some dimensions needed to the construction of that “last row”.

```

1298 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1299 {
1300   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1301     {
1302       \dim_gset:Nn \g_@@_ht_last_row_dim
1303         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1304       \dim_gset:Nn \g_@@_dp_last_row_dim
1305         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1306     }
1307 }

1308 \seq_gclear:N \g_@@_cols_vlism_seq
1309 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1310 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

The value of `\g_@@_pos_of_blocks_seq` has been written on the `aux` file and loaded before the (potential) execution of the `\CodeBefore`. Now, we clear that variable because it will be reconstructed during the creation of the array.

```

1311 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the `aux` file.

```

1312 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1313 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

The code in `\@@_pre_array_ii:` is used only here.

```

1314 \@@_pre_array_ii:

```

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

```

1315 \box_clear_new:N \l_@@_the_array_box

```

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

```

1316 \@@_construct_preamble:

```

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

We compute the width of both delimiters. We remember that, when the environment `{NiceArray}` is used, it's possible to specify the delimiters in the preamble (eg `[ccc]`).

```

1317 \dim_zero_new:N \l_@@_left_delim_dim
1318 \dim_zero_new:N \l_@@_right_delim_dim
1319 \bool_if:NTF \l_@@_NiceArray_bool
1320 {
1321   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1322   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1323 }
1324 {

```

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

```

1325 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1326 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1327 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1328 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1329 }

```

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).

```

1330 \hbox_set:Nw \l_@@_the_array_box
1331 \skip_horizontal:N \l_@@_left_margin_dim
1332 \skip_horizontal:N \l_@@_extra_left_margin_dim
1333 \c_math_toggle_token
1334 \bool_if:NTF \l_@@_light_syntax_bool
1335 { \use:c { @@-light-syntax } }
1336 { \use:c { @@-normal-syntax } }
1337 }

```

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

```

1338 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1339 {
1340   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1341   \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...

```

1342 \@@_pre_array:
1343 }

```

The `\CodeBefore`

The following command will be executed if the `\CodeBefore` has to be actually executed.

```

1344 \cs_new_protected:Npn \@@_pre_code_before:
1345 {

```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1346 \int_set:Nn \c@iRow { \seq_item:Nn \c_@@_size_seq 2 }
1347 \int_set:Nn \c@jCol { \seq_item:Nn \c_@@_size_seq 5 }
1348 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \c_@@_size_seq 3 }
1349 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \c_@@_size_seq 6 }

```

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.

```

1350 \pgfsys@markposition { \@@_env: - position }
1351 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1352 \pgfpicture
1353 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1354 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1355 {
1356   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1357   \pgfcoordinate { \@@_env: - row - ##1 }
1358   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1359 }

```

Now, the recreation of the `col` nodes.

```

1360 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1361 {
1362   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1363   \pgfcoordinate { \@@_env: - col - ##1 }
1364   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1365 }

```

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

```

1366 \@@_create_diag_nodes:

```

Now, the creation of the cell nodes (`i-j`), and, maybe also the “medium nodes” and the “large nodes”.

```

1367 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1368 \endpgfpicture

```

Now, the recreation of the nodes of the blocks *which have a name*.

```

1369 \@@_create_blocks_nodes:
1370 \bool_if:NT \c_@@_tikz_loaded_bool
1371 {
1372   \tikzset
1373   {
1374     every~picture / .style =
1375     { overlay , name~prefix = \@@_env: - }
1376   }
1377 }
1378 \cs_set_eq:NN \cellcolor \@@_cellcolor
1379 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1380 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1381 \cs_set_eq:NN \rowcolor \@@_rowcolor
1382 \cs_set_eq:NN \rowcolors \@@_rowcolors
1383 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1384 \cs_set_eq:NN \arraycolor \@@_arraycolor
1385 \cs_set_eq:NN \columncolor \@@_columncolor
1386 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1387 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1388 }

```

```

1389 \cs_new_protected:Npn \@@_exec_code_before:
1390 {
1391   \seq_gclear_new:N \g_@@_colors_seq
1392   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1393   \group_begin:

```

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

```

1394 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token

```

Here is the `\CodeBefore`. The construction is a bit complicated because `\l_@@_code_before_tl` may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do *not* execute the rest of `\l_@@_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That's why we begin with a `\q_stop`: it will be used to discard the rest of `\l_@@_code_before_tl`.

```
1395 \exp_last_unbraced:NV \@@_CodeBefore_keys: \l_@@_code_before_tl \q_stop
```

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

```
1396 \@@_actually_color:
1397 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1398 \group_end:
1399 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1400 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1401 }
```

```
1402 \keys_define:nn { NiceMatrix / CodeBefore }
1403 {
1404   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1405   create-cell-nodes .default:n = true ,
1406   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1407   sub-matrix .value_required:n = true ,
1408   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1409   delimiters / color .value_required:n = true ,
1410   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1411 }
1412 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1413 {
1414   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1415   \@@_CodeBefore:w
1416 }
```

We have extracted the options of the keyword `\CodeBefore` in order to see whether the key `create-cell-nodes` has been used. Now, you can execute the rest of the `\CodeAfter`, excepted, of course, if we are in the first compilation.

```
1417 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1418 {
1419   \bool_if:NT \g_@@_aux_found_bool
1420   {
1421     \@@_pre_code_before:
1422     #1
1423   }
1424 }
```

By default, if the user uses the `\CodeBefore`, only the `col` nodes, `row` nodes and `diag` nodes are available in that `\CodeBefore`. With the key `create-cell-nodes`, the cell nodes, that is to say the nodes of the form `(i-j)` (but not the extra nodes) are also available because those nodes also are recreated and that recreation is done by the following command.

```
1425 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1426 {
1427   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1428   {
1429     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1430     \pgfcoordinate { \@@_env: - row - ##1 - base }
1431     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1432     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1433     {
1434       \cs_if_exist:cT
1435       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
```

```

1436         {
1437             \pgfsys@getposition
1438             { \@@_env: - ##1 - ####1 - NW }
1439             \@@_node_position:
1440             \pgfsys@getposition
1441             { \@@_env: - ##1 - ####1 - SE }
1442             \@@_node_position_i:
1443             \@@_pgf_rect_node:nnn
1444             { \@@_env: - ##1 - ####1 }
1445             { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1446             { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1447         }
1448     }
1449 }
1450 \int_step_inline:nn \c@iRow
1451 {
1452     \pgfnodealias
1453     { \@@_env: - ##1 - last }
1454     { \@@_env: - ##1 - \int_use:N \c@jCol }
1455 }
1456 \int_step_inline:nn \c@jCol
1457 {
1458     \pgfnodealias
1459     { \@@_env: - last - ##1 }
1460     { \@@_env: - \int_use:N \c@iRow - ##1 }
1461 }
1462 \@@_create_extra_nodes:
1463 }

```

```

1464 \cs_new_protected:Npn \@@_create_blocks_nodes:
1465 {
1466     \pgfpicture
1467     \pgf@relevantforpicturesizefalse
1468     \pgfrememberpicturepositiononpagetrue
1469     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1470     { \@@_create_one_block_node:nnnnn ##1 }
1471     \endpgfpicture
1472 }

```

The following command is called `\@@_create_one_block_node:nnnnn` but, in fact, it creates a node only if the last argument (#5) which is the name of the block, is not empty.⁶⁵

```

1473 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1474 {
1475     \tl_if_empty:nF { #5 }
1476     {
1477         \@@_qpoint:n { col - #2 }
1478         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1479         \@@_qpoint:n { #1 }
1480         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1481         \@@_qpoint:n { col - \@@_succ:n { #4 } }
1482         \dim_set_eq:NN \l_tmpc_dim \pgf@x
1483         \@@_qpoint:n { \@@_succ:n { #3 } }
1484         \dim_set_eq:NN \l_tmpd_dim \pgf@y
1485         \@@_pgf_rect_node:nnnnn
1486         { \@@_env: - #5 }
1487         { \dim_use:N \l_tmpa_dim }
1488         { \dim_use:N \l_tmpb_dim }
1489         { \dim_use:N \l_tmpc_dim }
1490         { \dim_use:N \l_tmpd_dim }

```

⁶⁵Moreover, there is also in the list `\g_@@_pos_of_blocks_seq` the positions of the dotted lines (created by `\Cdots`, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).

```

1491     }
1492 }

1493 \cs_new_protected:Npn \@@_patch_for_revtext:
1494 {
1495   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1496   \cs_set_eq:NN \insert@column \insert@column@array
1497   \cs_set_eq:NN \@classx \@classx@array
1498   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1499   \cs_set_eq:NN \@arraycr \@arraycr@array
1500   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1501   \cs_set_eq:NN \array \array@array
1502   \cs_set_eq:NN \@array \@array@array
1503   \cs_set_eq:NN \@tabular \@tabular@array
1504   \cs_set_eq:NN \@mkpream \@mkpream@array
1505   \cs_set_eq:NN \endarray \endarray@array
1506   \cs_set:Npn \@tabarray { \@ifnextchar [ { \array } { \array [ c ] } }
1507   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1508 }

```

The environment {NiceArrayWithDelims}

```

1509 \NewDocumentEnvironment { NiceArrayWithDelims }
1510 { m m O { } m ! O { } t \CodeBefore }
1511 {
1512   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1513   \@@_provide_pgfsyspdfmark:
1514   \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.

```

1515   \bgroup

1516   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1517   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1518   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1519   \int_gzero:N \g_@@_block_box_int
1520   \dim_zero:N \g_@@_width_last_col_dim
1521   \dim_zero:N \g_@@_width_first_col_dim
1522   \bool_gset_false:N \g_@@_row_of_col_done_bool
1523   \str_if_empty:NT \g_@@_name_env_str
1524   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }

```

The following line will be deleted when we will consider that only versions of `siunitx` after v3.0 are compatible with `nicematrix`.

```

1525   \@@_adapt_S_column:
1526   \bool_if:NTF \l_@@_NiceTabular_bool
1527     \mode_leave_vertical:
1528     \@@_test_if_math_mode:
1529   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1530   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁶⁶. 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.

⁶⁶e.g. `\color[rgb]{0.5,0.5,0}`


```
1531 \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.

```
1532 \cs_if_exist:NT \tikz@library@external@loaded
1533 {
1534   \tikzexternaldisable
1535   \cs_if_exist:NT \ifstandalone
1536   { \tikzset { external / optimize = false } }
1537 }
```

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

```
1538 \int_gincr:N \g_@@_env_int
1539 \bool_if:NF \l_@@_block_auto_columns_width_bool
1540 { \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 (except the blocks with the key `hvlines`).

```
1541 \seq_gclear:N \g_@@_blocks_seq
1542 \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`.

```
1543 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1544 \seq_gclear:N \g_@@_pos_of_xdots_seq
1545 \tl_gclear_new:N \g_@@_code_before_tl
1546 \tl_gclear:N \g_@@_row_style_tl
```

We load all the informations written in the aux file during previous compilations corresponding to the current environment.

```
1547 \bool_gset_false:N \g_@@_aux_found_bool
1548 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1549 {
1550   \bool_gset_true:N \g_@@_aux_found_bool
1551   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1552 }
```

Now, we prepare the token list for the instructions that we will have to write on the aux file at the end of the environment.

```
1553 \tl_gclear:N \g_@@_aux_tl
1554 \tl_if_empty:NF \g_@@_code_before_tl
1555 {
1556   \bool_set_true:N \l_@@_code_before_bool
1557   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1558 }
```

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`.

```
1559 \bool_if:NTF \l_@@_NiceArray_bool
1560 { \keys_set:nn { NiceMatrix / NiceArray } }
1561 { \keys_set:nn { NiceMatrix / pNiceArray } }
1562 { #3 , #5 }

1563 \tl_if_empty:NF \l_@@_rules_color_tl
1564 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
```

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 body 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:.`

```

1565 \IfBooleanTF { #6 } \l_@@_pre_array_i:w \l_@@_pre_array:
1566 }
1567 {
1568 \bool_if:NTF \l_@@_light_syntax_bool
1569 { \use:c { end @@-light-syntax } }
1570 { \use:c { end @@-normal-syntax } }
1571 \c_math_toggle_token
1572 \skip_horizontal:N \l_@@_right_margin_dim
1573 \skip_horizontal:N \l_@@_extra_right_margin_dim
1574 \hbox_set_end:

```

End of the construction of the array (in the box `\l_@@_the_array_box`).

If the user has used the key `width` without any column `X`, we raise an error.

```

1575 \bool_if:NT \l_@@_width_used_bool
1576 {
1577 \int_compare:nNnT \g_@@_total_X_weight_int = 0
1578 { \@@_error:n { width~without~X~columns } }
1579 }

```

Now, if there is at least one `X`-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `\l_@@_X_columns_dim` will be the width of a column of weight 1. For a `X`-column of weight n , the width will be `\l_@@_X_columns_dim` multiplied by n .

```

1580 \int_compare:nNnT \g_@@_total_X_weight_int > 0
1581 {
1582 \tl_gput_right:Nx \g_@@_aux_tl
1583 {
1584 \bool_set_true:N \l_@@_X_columns_aux_bool
1585 \dim_set:Nn \l_@@_X_columns_dim
1586 {
1587 \dim_compare:nNnTF
1588 {
1589 \dim_abs:n
1590 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1591 }
1592 <
1593 { 0.001 pt }
1594 { \dim_use:N \l_@@_X_columns_dim }
1595 {
1596 \dim_eval:n
1597 {
1598 ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1599 / \int_use:N \g_@@_total_X_weight_int
1600 + \l_@@_X_columns_dim
1601 }
1602 }
1603 }
1604 }
1605 }

```

If the user has used the key `last-row` with a value, we control that the given value is correct (since we have just constructed the array, we know the real number of rows of the array).

```

1606 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1607 {
1608 \bool_if:NF \l_@@_last_row_without_value_bool
1609 {
1610 \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1611 {
1612 \@@_error:n { Wrong~last~row }
1613 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1614 }
1615 }
1616 }

```

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”.⁶⁷

```

1617 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1618 \bool_if:nTF \g_@@_last_col_found_bool
1619 { \int_gdecr:N \c@jCol }
1620 {
1621   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1622   { \@@_error:n { last~col~not~used } }
1623 }

```

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

```

1624 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1625 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 126).

```

1626 \int_compare:nNnT \l_@@_first_col_int = 0
1627 {
1628   \skip_horizontal:N \col@sep
1629   \skip_horizontal:N \g_@@_width_first_col_dim
1630 }

```

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.

```

1631 \bool_if:nTF \l_@@_NiceArray_bool
1632 {
1633   \str_case:VnF \l_@@_baseline_tl
1634   {
1635     b \@@_use_arraybox_with_notes_b:
1636     c \@@_use_arraybox_with_notes_c:
1637   }
1638   \@@_use_arraybox_with_notes:
1639 }

```

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).

```

1640 {
1641   \int_compare:nNnTF \l_@@_first_row_int = 0
1642   {
1643     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1644     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1645   }
1646   { \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”.⁶⁸

```

1647   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1648   {
1649     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1650     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1651   }
1652   { \dim_zero:N \l_tmpb_dim }
1653   \hbox_set:Nn \l_tmpa_box
1654   {
1655     \c_math_toggle_token
1656     \tl_if_empty:NF \l_@@_delimiters_color_tl
1657     { \color { \l_@@_delimiters_color_tl } }

```

⁶⁷We remind that the potential “first column” (exterior) has the number 0.

⁶⁸A 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).

```

1658         \exp_after:wN \left \g_@@_left_delim_tl
1659         \vcenter
1660         {

```

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.

```

1661         \skip_vertical:N -\l_tmpa_dim
1662         \hbox
1663         {
1664             \bool_if:NTF \l_@@_NiceTabular_bool
1665             { \skip_horizontal:N -\tabcolsep }
1666             { \skip_horizontal:N -\arraycolsep }
1667             \@@_use_arraybox_with_notes_c:
1668             \bool_if:NTF \l_@@_NiceTabular_bool
1669             { \skip_horizontal:N -\tabcolsep }
1670             { \skip_horizontal:N -\arraycolsep }
1671         }

```

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

```

1672         \skip_vertical:N -\l_tmpb_dim
1673     }

```

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.

```

1674         \tl_if_empty:NF \l_@@_delimiters_color_tl
1675         { \color { \l_@@_delimiters_color_tl } }
1676         \exp_after:wN \right \g_@@_right_delim_tl
1677         \c_math_toggle_token
1678     }

```

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.

```

1679         \bool_if:NTF \l_@@_delimiters_max_width_bool
1680         {
1681             \@@_put_box_in_flow_bis:nn
1682             \g_@@_left_delim_tl \g_@@_right_delim_tl
1683         }
1684         \@@_put_box_in_flow:
1685     }

```

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. 127).

```

1686         \bool_if:NT \g_@@_last_col_found_bool
1687         {
1688             \skip_horizontal:N \g_@@_width_last_col_dim
1689             \skip_horizontal:N \col@sep
1690         }
1691         \bool_if:NF \l_@@_Matrix_bool
1692         {
1693             \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1694             { \@@_error:n { columns-not-used } }
1695         }
1696         \group_begin:
1697         \globaldefs = 1
1698         \@@_msg_redirect_name:nn { columns-not-used } { error }
1699         \group_end:
1700         \@@_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.

```

1701     \egroup

```

We want to write on the aux file all the informations corresponding to the current environment.

```

1702     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1703     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }

```

```

1704 \iow_now:Nx \@mainaux
1705 {
1706     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ t1 }
1707     { \exp_not:V \g_@@_aux_t1 }
1708 }
1709 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1710 \bool_if:NT \c_@@_footnote_bool \endsavenotes
1711 }

```

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

We construct the preamble of the array

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

The preamble given by the final user is in `\g_@@_preamble_t1` and the modified version will be stored in `\g_@@_preamble_t1` also.

```

1712 \cs_new_protected:Npn \@_construct_preamble:
1713 {

```

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 the L3 programming layer.

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.

```

1714 \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).

```

1715 \bool_if:NF \l_@@_Matrix_bool
1716 {
1717     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1718     \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

If the package `varwidth` has defined the column type `V`, we protect from expansion by redefining it to `\@@_V:` (which will be caught by our system).

```

1719 \cs_if_exist:NT \NC@find@V { \@@_newcolumntype V { \@@_V: } }

```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by the L3 programming layer).

```

1720 \exp_args:NV \@temptokena \g_@@_preamble_t1

```

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

```

1721 \@tempswatrue

```

The following line actually does the expansion (it’s has been copied from `array.sty`). The expanded version is still in `\@temptokena`.

```

1722 \@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`.

```

1723     \int_gzero:N \c@jCol
1724     \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1725     \bool_gset_false:N \g_tmpb_bool
1726     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1727     {
1728         \tl_gset:Nn \g_@@_preamble_tl
1729         { ! { \skip_horizontal:N \arrayrulewidth } }
1730     }
1731     {
1732         \clist_if_in:NnT \l_@@_vlines_clist 1
1733         {
1734             \tl_gset:Nn \g_@@_preamble_tl
1735             { ! { \skip_horizontal:N \arrayrulewidth } }
1736         }
1737     }

```

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`).

```

1738     \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1739     \int_zero:N \l_tmpa_int

```

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

```

1740     \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1741     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1742 }

```

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

```

1743     \bool_if:NT \l_@@_colortbl_like_bool
1744     {
1745         \regex_replace_all:NnN
1746         \c_@@_columncolor_regex
1747         { \c { @@_columncolor_preamble } }
1748         \g_@@_preamble_tl
1749     }

```

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

```

1750     \group_end:

```

If there was delimiters at the beginning or at the end of the preamble, the environment `{NiceArray}` is transformed into an environment `{xNiceMatrix}`.

```

1751     \bool_lazy_or:nnT
1752     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1753     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1754     { \bool_set_false:N \l_@@_NiceArray_bool }

```

We want to remind whether there is a specifier `|` at the end of the preamble.

```

1755     \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }

```

We complete the preamble with the potential “exterior columns” (on both sides).

```

1756     \int_compare:nNnTF \l_@@_first_col_int = 0
1757     { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1758     {
1759         \bool_lazy_all:nT

```

```

1760     {
1761         \l_@@_NiceArray_bool
1762         { \bool_not_p:n \l_@@_NiceTabular_bool }
1763         { \tl_if_empty_p:N \l_@@_vlines_clist }
1764         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1765     }
1766     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1767 }
1768 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1769 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1770 {
1771     \bool_lazy_all:nT
1772     {
1773         \l_@@_NiceArray_bool
1774         { \bool_not_p:n \l_@@_NiceTabular_bool }
1775         { \tl_if_empty_p:N \l_@@_vlines_clist }
1776         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1777     }
1778     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1779 }

```

We add a last column to raise a good error message when the user puts 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`).

```

1780     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1781     {
1782         \tl_gput_right:Nn \g_@@_preamble_tl
1783         { > { \@@_error_too_much_cols: } 1 }
1784     }
1785 }

```

```

1786 \cs_new_protected:Npn \@@_patch_preamble:n #1
1787 {
1788     \str_case:nnF { #1 }
1789     {
1790         c { \@@_patch_preamble_i:n #1 }
1791         l { \@@_patch_preamble_i:n #1 }
1792         r { \@@_patch_preamble_i:n #1 }
1793         > { \@@_patch_preamble_ii:nn #1 }
1794         ! { \@@_patch_preamble_ii:nn #1 }
1795         @ { \@@_patch_preamble_ii:nn #1 }
1796         | { \@@_patch_preamble_iii:n #1 }
1797         p { \@@_patch_preamble_iv:n #1 }
1798         b { \@@_patch_preamble_iv:n #1 }
1799         m { \@@_patch_preamble_iv:n #1 }
1800         \@@_V: { \@@_patch_preamble_v:n }
1801         V { \@@_patch_preamble_v:n }
1802         \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
1803         \@@_W: { \@@_patch_preamble_vi:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1804         \@@_S: { \@@_patch_preamble_vii:n }
1805         ( { \@@_patch_preamble_viii:nn #1 }
1806         [ { \@@_patch_preamble_viii:nn #1 }
1807         \{ { \@@_patch_preamble_viii:nn #1 }
1808         ) { \@@_patch_preamble_ix:nn #1 }
1809         ] { \@@_patch_preamble_ix:nn #1 }
1810         \} { \@@_patch_preamble_ix:nn #1 }
1811         X { \@@_patch_preamble_x:n }

```

When `tabularx` is loaded, a local redefinition of the specifier ‘X’ is done to replace ‘X’ by ‘@_X’. Thus, our column type ‘X’ will be used in the ‘NiceTabularX’.

```

1812     \@@_X { \@@_patch_preamble_x:n }
1813     \q_stop { }
1814 }

```

```

1815 {
1816   \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1817   { \@@_patch_preamble_xii:n #1 }
1818   {
1819     \str_if_eq:VnTF \l_@@_letter_vlism_tl { #1 }
1820     {
1821       \seq_gput_right:Nx \g_@@_cols_vlism_seq
1822       { \int_eval:n { \c@jCol + 1 } }
1823       \tl_gput_right:Nx \g_@@_preamble_tl
1824       { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1825       \@@_patch_preamble:n
1826     }
1827     {
1828       \bool_lazy_and:nnTF
1829       { \str_if_eq_p:nn { : } { #1 } }
1830       \c_@@_arydshln_loaded_bool
1831       {
1832         \tl_gput_right:Nn \g_@@_preamble_tl { : }
1833         \@@_patch_preamble:n
1834       }
1835       { \@@_fatal:nn { unknown~column~type } { #1 } }
1836     }
1837   }
1838 }
1839 }

```

For c, l and r

```

1840 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1841 {
1842   \tl_gput_right:Nn \g_@@_preamble_tl
1843   {
1844     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
1845     #1
1846     < \@@_cell_end:
1847   }

```

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

```

1848   \int_gincr:N \c@jCol
1849   \@@_patch_preamble_xi:n
1850 }

```

For >, ! and @

```

1851 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1852 {
1853   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1854   \@@_patch_preamble:n
1855 }

```

For |

```

1856 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1857 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1858   \int_incr:N \l_tmpa_int
1859   \@@_patch_preamble_iii_i:n
1860 }

```

```

1861 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1862 {
1863   \str_if_eq:nnTF { #1 } |
1864   { \@@_patch_preamble_iii:n | }
1865   {
1866     \tl_gput_right:Nx \g_@@_preamble_tl
1867     {

```



```

1868         \exp_not:N !
1869         {
1870             \skip_horizontal:n
1871             {
1872                 \dim_eval:n
1873                 {
1874                     \arrayrulewidth * \l_tmpa_int
1875                     + \doublerulesep * ( \l_tmpa_int - 1)
1876                 }
1877             }
1878         }
1879     }
1880     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1881     {
1882         \@@_vline:nnnn
1883         { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } { 1 } { }
1884     }
1885     \int_zero:N \l_tmpa_int
1886     \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
1887     \@@_patch_preamble:n #1
1888 }
1889 }
1890 \bool_new:N \l_@@_bar_at_end_of_pream_bool

```

The specifier `p` (and also the specifiers `m` and `b`) have an optional argument between square brackets for a list of *key-value* pairs. Here are the corresponding keys. This set of keys will also be used by the `X` columns.

```

1891 \keys_define:nn { WithArrows / p-column }
1892 {
1893     r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
1894     r .value_forbidden:n = true ,
1895     c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
1896     c .value_forbidden:n = true ,
1897     l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
1898     l .value_forbidden:n = true ,
1899     si .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
1900     si .value_forbidden:n = true ,
1901     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
1902     p .value_forbidden:n = true ,
1903     t .meta:n = p ,
1904     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
1905     m .value_forbidden:n = true ,
1906     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
1907     b .value_forbidden:n = true ,
1908 }

```

For `p`, `b` and `m`. The argument `#1` is that value : `p`, `b` or `m`.

```

1909 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
1910 {
1911     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

Now, you look for a potential character [after the letter of the specifier (for the options).

```

1912     \@@_patch_preamble_iv_i:n
1913 }
1914 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
1915 {
1916     \str_if_eq:nnTF { #1 } { [ ]
1917         { \@@_patch_preamble_iv_ii:w [ ]
1918           { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
1919         }
1920     \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
1921     { \@@_patch_preamble_iv_iii:nn { #1 } }

```

#1 is the optional argument of the specifier (a list of *key-value* pairs).
 #2 is the mandatory argument of the specifier: the width of the column.

```
1922 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
1923 {
```

The possible values of `\l_@@_hpos_col_str` are j (for *justified* which is the initial value), l, c and r (when the user has used the corresponding key in the optional argument of the specifier).

```
1924   \str_set:Nn \l_@@_hpos_col_str { j }
1925   \keys_set:nn { WithArrows / p-column } { #1 }
1926   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
1927 }
```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`.

```
1928 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
1929 {
1930   \use:x
1931   {
1932     \@@_patch_preamble_iv_v:nnnnnnn
1933     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
1934     { \dim_eval:n { #1 } }
1935   }
```

The parameter `\l_@@_hpos_col_str` (as `\l_@@_vpos_col_str`) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter `\l_@@_hpos_cell_str` which will provide the horizontal alignment of the column to which belongs the cell.

```
1936       \str_if_eq:VnTF \l_@@_hpos_col_str j
1937       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
1938       {
1939         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
1940         { \l_@@_hpos_col_str }
1941       }
1942       \str_case:Vn \l_@@_hpos_col_str
1943       {
1944         c { \exp_not:N \centering }
1945         l { \exp_not:N \raggedright }
1946         r { \exp_not:N \raggedleft }
1947       }
1948     }
1949     { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
1950     { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
1951     { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
1952     { #2 }
1953 }
```

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

```
1954   \int_gincr:N \c@jCol
1955   \@@_patch_preamble_xi:n
1956 }
```

#1 is the optional argument of `{minipage}`: t of b. Indeed, for the columns of type m, we use the value b here because there is a special post-action in order to center vertically the box (see #4).

#2 is the width of the `{minipage}`, that is to say also the width of the column.

#3 is the coding for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing). It's also possible to put in that #3 some code to fix the value of `\l_@@_hpos_cell_str` which will be available in each cell of the column.

#4 is an extra-code which contains `\@@_center_cell_box:` (when the column is a m column) or nothing (in the other cases).

#5 is a code put just before the c.

#6 is a code put just after the c.

#7 is the type of environment: `minipage` or `varwidth`.

```
1957 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7
```

```

1958 {
1959   \tl_gput_right:Nn \g_@@_preamble_tl
1960   {
1961     > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

```

1962       \dim_set:Nn \l_@@_col_width_dim { #2 }
1963       \@@_cell_begin:w
1964       \begin { #7 } [ #1 ] { #2 }

```

The following lines have been taken from `array.sty`.

```

1965       \everypar
1966       {
1967         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
1968         \everypar { }
1969       }

```

Now, the potential code for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing).

```

1970       #3

```

The following code is to allow something like `\centering` in `\RowStyle`.

```

1971       \g_@@_row_style_tl
1972       \arraybackslash
1973       #5
1974     }
1975     c
1976     < {
1977       #6

```

The following line has been taken from `array.sty`.

```

1978       \@finalstrut \@arstrutbox
1979       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
1980       \end { #7 }

```

If the letter in the preamble is `m`, `#3` will be equal to `\@@_center_cell_box:` (see just below).

```

1981       #4
1982       \@@_cell_end:
1983     }
1984   }
1985 }

```

The following command will be used in `m-columns` in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in `array`) that if the height of the cell is no more that the height of `\@arstrutbox`, there is only one row.

```

1986 \cs_new_protected:Npn \@@_center_cell_box:
1987 {

```

By putting instructions in `\g_@@_post_action_cell_tl`, we require a post-action of the box `\l_@@_cell_box`.

```

1988   \tl_gput_right:Nn \g_@@_post_action_cell_tl
1989   {
1990     \int_compare:nNnT
1991     { \box_ht:N \l_@@_cell_box }
1992     >
1993     { \box_ht:N \@arstrutbox }
1994     {
1995       \hbox_set:Nn \l_@@_cell_box
1996       {
1997         \box_move_down:nn
1998         {
1999           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox

```

```

2000             + \baselineskip ) / 2
2001         }
2002         { \box_use:N \l_@@_cell_box }
2003     }
2004 }
2005 }
2006 }

```

For V (similar to the V of varwidth).

```

2007 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2008 {
2009     \str_if_eq:nnTF { #1 } { [ ] }
2010     { \@@_patch_preamble_v_i:w [ ] }
2011     { \@@_patch_preamble_v_i:w [ ] { #1 } }
2012 }
2013 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2014 { \@@_patch_preamble_v_ii:nn { #1 } }
2015 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2016 {
2017     \str_set:Nn \l_@@_vpos_col_str { p }
2018     \str_set:Nn \l_@@_hpos_col_str { j }
2019     \keys_set:nn { WithArrows / p-column } { #1 }
2020     \bool_if:NTF \c_@@_varwidth_loaded_bool
2021     { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2022     {
2023         \@@_error:n { varwidth~not~loaded }
2024         \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2025     }
2026 }

```

For w and W

```

2027 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2028 {
2029     \tl_gput_right:Nn \g_@@_preamble_tl
2030     {
2031         > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

```

2032         \dim_set:Nn \l_@@_col_width_dim { #4 }
2033         \hbox_set:Nw \l_@@_cell_box
2034         \@@_cell_begin:w
2035         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2036     }
2037     c
2038     < {
2039         \@@_cell_end:
2040         #1
2041         \hbox_set_end:
2042         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2043         \@@_adjust_size_box:
2044         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2045     }
2046 }

```

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

```

2047     \int_gincr:N \c@jCol
2048     \@@_patch_preamble_xi:n
2049 }

```

For `\@@_S:`. If the user has used `S[...]`, `S` has been replaced by `\@@_S:` during the first expansion of the preamble (done with the tools of standard LaTeX and array).

```

2050 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2051 {

```

```

2052 \str_if_eq:nnTF { #1 } { [ ]
2053 { \@@_patch_preamble_vii_i:w [ ]
2054 { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2055 }
2056 \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2057 { \@@_patch_preamble_vii_ii:n { #1 } }

```

For version of siunitx at least equal to 3.0, the adaptation is different from previous ones. We test the version of siunitx by the existence of the control sequence `\siunitx_cell_begin:w`. When we will decide that only the previous posterior to 3.0 are supported by nicematrix, we will delete the second definition of `\@@_patch_preamble_vii_ii:n`.

```

2058 \AtBeginDocument
2059 {
2060 \cs_if_exist:NTF \siunitx_cell_begin:w
2061 {
2062 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2063 {
2064 \tl_gput_right:Nn \g_@@_preamble_tl
2065 {
2066 > {
2067 \@@_cell_begin:w
2068 \keys_set:nn { siunitx } { #1 }
2069 \siunitx_cell_begin:w
2070 }
2071 c
2072 < { \siunitx_cell_end: \@@_cell_end: }
2073 }

```

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

```

2074 \int_gincr:N \c@jCol
2075 \@@_patch_preamble_xi:n
2076 }
2077 }
2078 {
2079 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2080 {
2081 \tl_gput_right:Nn \g_@@_preamble_tl
2082 {
2083 > { \@@_cell_begin:w \c_@@_table_collect_begin_tl S { #1 } }
2084 c
2085 < { \c_@@_table_print_tl \@@_cell_end: }
2086 }
2087 \int_gincr:N \c@jCol
2088 \@@_patch_preamble_xi:n
2089 }
2090 }
2091 }

```

For `(`, `[` and `\{`.

```

2092 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2093 {
2094 \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }

```

If we are before the column 1 and not in `{NiceArray}`, we reserve space for the left delimiter.

```

2095 \int_compare:nNnTF \c@jCol = \c_zero_int
2096 {
2097 \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2098 {

```

In that case, in fact, the first letter of the preamble must be considered as the left delimiter of the array.

```

2099 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2100 \tl_gset:Nn \g_@@_right_delim_tl { . }
2101 \@@_patch_preamble:n #2

```

```

2102     }
2103     {
2104         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2105         \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2106     }
2107 }
2108 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2109 }
2110 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2111 {
2112     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2113     { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
2114     \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
2115     {
2116         \@@_error:nn { delimiter~after~opening } { #2 }
2117         \@@_patch_preamble:n
2118     }
2119     { \@@_patch_preamble:n #2 }
2120 }

```

For `)`, `]` and `\}`. We have two arguments for the following command because we directly read the following letter in the preamble (we have to see whether we have a opening delimiter following and we also have to see whether we are at the end of the preamble because, in that case, our letter must be considered as the right delimiter of the environment if the environment is `{NiceArray}`).

```

2121 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2122 {
2123     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2124     \tl_if_in:nnTF { ) ] \} } { #2 }
2125     { \@@_patch_preamble_ix_i:nnn #1 #2 }
2126     {
2127         \tl_if_eq:nnTF { \q_stop } { #2 }
2128         {
2129             \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2130             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2131             {
2132                 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2133                 \tl_gput_right:Nx \g_@@_internal_code_after_tl
2134                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2135                 \@@_patch_preamble:n #2
2136             }
2137         }
2138         {
2139             \tl_if_in:nnT { ( [ \{ } { #2 }
2140             { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2141             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2142             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2143             \@@_patch_preamble:n #2
2144         }
2145     }
2146 }
2147 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2148 {
2149     \tl_if_eq:nnTF { \q_stop } { #3 }
2150     {
2151         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2152         {
2153             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2154             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2155             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2156             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2157         }
2158         {

```

```

2159         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2160         \tl_gput_right:Nx \g_@@_internal_code_after_tl
2161         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2162         \@@_error:nn { double~closing~delimiter } { #2 }
2163     }
2164 }
2165 {
2166     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2167     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2168     \@@_error:nn { double~closing~delimiter } { #2 }
2169     \@@_patch_preamble:n #3
2170 }
2171 }

```

For the case of a letter X. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a [after the letter X.

```

2172 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2173 {
2174     \str_if_eq:nnTF { #1 } { [ ]
2175     { \@@_patch_preamble_x_i:w [ ]
2176     { \@@_patch_preamble_x_i:w [ ] #1 }
2177     }
2178 \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2179 { \@@_patch_preamble_x_ii:n { #1 } }

```

#1 is the optional argument of the X specifier (a list of *key-value* pairs).

The following set of keys is for the specifier X in the preamble of the array. Such specifier may have as keys all the keys of { WithArrows / p-column } but also a key as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter \l_@@_weight_int).

```

2180 \keys_define:nn { WithArrows / X-column }
2181 { unknown .code:n = \int_set:Nn \l_@@_weight_int { \l_keys_key_str } }

```

In the following command, #1 is the list of the options of the specifier X.

```

2182 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2183 {

```

The possible values of \l_@@_hpos_col_str are j (for *justified* which is the initial value), l, c and r (when the user has used the corresponding key in the optional argument of the specifier X).

```

2184     \str_set:Nn \l_@@_hpos_col_str { j }

```

The possible values of \l_@@_vpos_col_str are p (the initial value), m and b (when the user has used the corresponding key in the optional argument of the specifier X).

```

2185     \tl_set:Nn \l_@@_vpos_col_str { p }

```

The integer \l_@@_weight_int will be the weight of the X column (the initial value is 1). The user may specify a different value (such as 2, 3, etc.) by putting that value in the optional argument of the specifier. The weights of the X columns are used in the computation of the actual width of those columns as in tabu of tabularray.

```

2186     \int_zero_new:N \l_@@_weight_int
2187     \int_set:Nn \l_@@_weight_int { 1 }
2188     \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
2189     \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2190     \int_compare:nNnT \l_@@_weight_int < 0
2191     {
2192         \exp_args:Nnx \@@_error:nn { negative~weight }
2193         { \int_use:N \l_@@_weight_int }
2194         \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2195     }
2196     \int_gadd:Nn \g_@@_total_X_weight_int \l_@@_weight_int

```

We test whether we know the width of the X-columns by reading the `aux` file (after the first compilation, the width of the X-columns is computed and written in the `aux` file).

```

2197 \bool_if:NTF \l_@@_X_columns_aux_bool
2198 {
2199   \@@_patch_preamble_iv_iv:nn
2200   { \l_@@_weight_int \l_@@_X_columns_dim }
2201   { minipage }
2202 }
2203 {
2204   \tl_gput_right:Nn \g_@@_preamble_tl
2205   {
2206     > {
2207       \@@_cell_begin:w
2208       \bool_set_true:N \l_@@_X_column_bool

```

The following code will nullify the box of the cell.

```

2209       \tl_gput_right:Nn \g_@@_post_action_cell_tl
2210       { \hbox_set:Nn \l_@@_cell_box { } }

```

We put a `{minipage}` to give to the user the ability to put a command such as `\centering` in the `\RowStyle`.

```

2211       \begin { minipage } { 5 cm } \arraybackslash
2212     }
2213     c
2214     < {
2215       \end { minipage }
2216       \@@_cell_end:
2217     }
2218   }
2219   \int_gincr:N \c@jCol
2220   \@@_patch_preamble_xi:n
2221 }
2222 }

```

```

2223 \cs_new_protected:Npn \@@_patch_preamble_xii:n #1
2224 {
2225   \tl_gput_right:Nn \g_@@_preamble_tl
2226   { ! { \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`.

```

2227   \tl_gput_right:Nx \g_@@_internal_code_after_tl
2228   { \@@_vdottedline:n { \int_use:N \c@jCol } }
2229   \@@_patch_preamble:n
2230 }

```

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.

```

2231 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2232 {
2233   \str_if_eq:nnTF { #1 } { < }
2234   \@@_patch_preamble_xiii:n
2235   {
2236     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2237     {
2238       \tl_gput_right:Nn \g_@@_preamble_tl
2239       { ! { \skip_horizontal:N \arrayrulewidth } }
2240     }
2241     {
2242       \exp_args:NNx
2243       \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2244       {
2245         \tl_gput_right:Nn \g_@@_preamble_tl

```



```

2246         { ! { \skip_horizontal:N \arrayrulewidth } }
2247     }
2248 }
2249 \@@_patch_preamble:n { #1 }
2250 }
2251 }
2252 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2253 {
2254     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2255     \@@_patch_preamble_xi:n
2256 }

```

The redefinition of `\multicolumn`

The following command must *not* be protected since it begins with `\multispan` (a TeX primitive).

```

2257 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2258 {

```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```

2259     \multispan { #1 }
2260     \begingroup
2261     \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

You do the expansion of the (small) preamble with the tools of `array`.

```

2262     \@temptokena = { #2 }
2263     \@tempswatrue
2264     \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

Now, we patch the (small) preamble as we have done with the main preamble of the array.

```

2265     \tl_gclear:N \g_@@_preamble_tl
2266     \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

The following lines are an adaptation of the definition of `\multicolumn` in `array`.

```

2267     \exp_args:NV \mkpream \g_@@_preamble_tl
2268     \addtopreamble \empty
2269     \endgroup

```

Now, you do a treatment specific to `nicematrix` which has no equivalent in the original definition of `\multicolumn`.

```

2270     \int_compare:nNnT { #1 } > 1
2271     {
2272         \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2273         { \int_use:N \c@iRow - \@@_succ:n \c@jCol }
2274         \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2275         \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2276         {
2277             { \int_use:N \c@iRow }
2278             { \int_eval:n { \c@jCol + 1 } }
2279             { \int_use:N \c@iRow }
2280             { \int_eval:n { \c@jCol + #1 } }
2281             { } % for the name of the block
2282         }
2283     }

```

The following lines were in the original definition of `\multicolumn`.

```

2284     \cs_set:Npn \@sharp { #3 }
2285     \@arstrut
2286     \@preamble
2287     \null

```

We add some lines.

```

2288 \int_gadd:Nn \c@jCol { #1 - 1 }
2289 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2290 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2291 \ignorespaces
2292 }

```

The following commands will patch the (small) preamble of the `\multicolumn`. All those commands have a `m` in their name to recall that they deal with the redefinition of `\multicolumn`.

```

2293 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2294 {
2295   \str_case:nnF { #1 }
2296   {
2297     c { \@@_patch_m_preamble_i:n #1 }
2298     l { \@@_patch_m_preamble_i:n #1 }
2299     r { \@@_patch_m_preamble_i:n #1 }
2300     > { \@@_patch_m_preamble_ii:nn #1 }
2301     ! { \@@_patch_m_preamble_ii:nn #1 }
2302     @ { \@@_patch_m_preamble_ii:nn #1 }
2303     | { \@@_patch_m_preamble_iii:n #1 }
2304     p { \@@_patch_m_preamble_iv:nnn t #1 }
2305     m { \@@_patch_m_preamble_iv:nnn c #1 }
2306     b { \@@_patch_m_preamble_iv:nnn b #1 }
2307     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2308     \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2309     \q_stop { }
2310   }
2311   { \@@_fatal:nn { unknown~column~type } { #1 } }
2312 }

```

For `c`, `l` and `r`

```

2313 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2314 {
2315   \tl_gput_right:Nn \g_@@_preamble_tl
2316   {
2317     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2318     #1
2319     < \@@_cell_end:
2320   }

```

We test for the presence of a `<`.

```

2321 \@@_patch_m_preamble_x:n
2322 }

```

For `>`, `!` and `@`

```

2323 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2324 {
2325   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2326   \@@_patch_m_preamble:n
2327 }

```

For `|`

```

2328 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2329 {
2330   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2331   \@@_patch_m_preamble:n
2332 }

```

For `p`, `m` and `b`

```

2333 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2334 {
2335   \tl_gput_right:Nn \g_@@_preamble_tl

```

```

2336 {
2337   > {
2338     \@@_cell_begin:w
2339     \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2340     \mode_leave_vertical:
2341     \arraybackslash
2342     \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2343   }
2344   c
2345   < {
2346     \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2347     \end { minipage }
2348     \@@_cell_end:
2349   }
2350 }

```

We test for the presence of a <.

```

2351 \@@_patch_m_preamble_x:n
2352 }

```

For w and W

```

2353 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2354 {
2355   \tl_gput_right:Nn \g_@@_preamble_tl
2356   {
2357     > {
2358       \hbox_set:Nw \l_@@_cell_box
2359       \@@_cell_begin:w
2360       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2361     }
2362     c
2363     < {
2364       \@@_cell_end:
2365       #1
2366       \hbox_set_end:
2367       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2368       \@@_adjust_size_box:
2369       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2370     }
2371   }

```

We test for the presence of a <.

```

2372 \@@_patch_m_preamble_x:n
2373 }

```

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 vlins is used.

```

2374 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2375 {
2376   \str_if_eq:nnTF { #1 } { < }
2377   \@@_patch_m_preamble_ix:n
2378   {
2379     \tl_if_eq:NnTF \l_@@_vlins_clist { all }
2380     {
2381       \tl_gput_right:Nn \g_@@_preamble_tl
2382       { ! { \skip_horizontal:N \arrayrulewidth } }
2383     }
2384     {
2385       \exp_args:NNx
2386       \clist_if_in:NnT \l_@@_vlins_clist { \@@_succ:n \c@jCol }
2387       {
2388         \tl_gput_right:Nn \g_@@_preamble_tl
2389         { ! { \skip_horizontal:N \arrayrulewidth } }
2390       }
2391     }

```

```

2392     \@@_patch_m_preamble:n { #1 }
2393   }
2394 }
2395 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2396 {
2397   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2398   \@@_patch_m_preamble_x:n
2399 }

```

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`).

```

2400 \cs_new_protected:Npn \@@_put_box_in_flow:
2401 {
2402   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2403   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2404   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2405     { \box_use_drop:N \l_tmpa_box }
2406     \@@_put_box_in_flow_i:
2407 }

```

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).

```

2408 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2409 {
2410   \pgfpicture
2411     \@@_qpoint:n { row - 1 }
2412     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2413     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
2414     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2415     \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).

```

2416   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2417   {
2418     \int_set:Nn \l_tmpa_int
2419     {
2420       \str_range:Nnn
2421         \l_@@_baseline_tl
2422         6
2423         { \tl_count:V \l_@@_baseline_tl }
2424     }
2425     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2426   }
2427   {
2428     \str_case:VnF \l_@@_baseline_tl
2429     {
2430       { t } { \int_set:Nn \l_tmpa_int 1 }
2431       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2432     }
2433     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2434     \bool_lazy_or:nnT
2435     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2436     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2437     {
2438       \@@_error:n { bad~value~for~baseline }
2439       \int_set:Nn \l_tmpa_int 1
2440     }
2441     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2442         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2443     }
2444     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2445     \endpgfpicture
2446     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2447     \box_use_drop:N \l_tmpa_box
2448 }

```

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).

```

2449 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2450 {

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

2451     \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2452     {
2453         \box_set_wd:Nn \l_@@_the_array_box
2454         { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2455     }

```

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

```

2456     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }

```

The `\hbox` avoids that the `pgfpicture` inside `\@@_draw_blocks` adds a extra vertical space before the notes.

```

2457     \hbox
2458     {
2459         \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.

```

2460         \@@_create_extra_nodes:
2461         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2462     }
2463     \bool_lazy_or:nnT
2464     { \int_compare_p:nNn \c@tabularnote > 0 }
2465     { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2466     \@@_insert_tabularnotes:
2467     \end { minipage }
2468 }

2469 \cs_new_protected:Npn \@@_insert_tabularnotes:
2470 {
2471     \skip_vertical:N 0.65ex

```

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

```

2472     \group_begin:
2473     \l_@@_notes_code_before_tl
2474     \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.

```

2475     \int_compare:nNnT \c@tabularnote > 0
2476     {
2477         \bool_if:NTF \l_@@_notes_para_bool
2478         {

```

```

2479         \begin { tabularnotes* }
2480         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2481         \end { tabularnotes* }

```

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

```

2482         \par
2483     }
2484     {
2485         \tabularnotes
2486         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2487         \endtabularnotes
2488     }
2489 }
2490 \unskip
2491 \group_end:
2492 \bool_if:NT \l_@@_notes_bottomrule_bool
2493 {
2494     \bool_if:NTF \c_@@_booktabs_loaded_bool
2495     {

```

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

```

2496         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2497         { \CT@arc@ \hrule height \heavyrulewidth }
2498     }
2499     { \@@_error:n { bottomrule~without~booktabs } }
2500 }
2501 \l_@@_notes_code_after_tl
2502 \seq_gclear:N \g_@@_tabularnotes_seq
2503 \int_gzero:N \c@tabularnote
2504 }

```

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`.

```

2505 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2506 {
2507     \pgfpicture
2508     \@@_qpoint:n { row - 1 }
2509     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2510     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2511     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2512     \endpgfpicture
2513     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2514     \int_compare:nNnT \l_@@_first_row_int = 0
2515     {
2516         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2517         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2518     }
2519     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2520 }

```

Now, the general case.

```

2521 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2522 {

```

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

```

2523     \tl_if_eq:NnT \l_@@_baseline_tl { t }
2524     { \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`.

```

2525 \pgfpicture
2526 \@@_qpoint:n { row - 1 }
2527 \dim_gset_eq:NW \g_tmpa_dim \pgf@y
2528 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2529 {
2530   \int_set:Nn \l_tmpa_int
2531   {
2532     \str_range:Nnn
2533       \l_@@_baseline_tl
2534       6
2535     { \tl_count:V \l_@@_baseline_tl }
2536   }
2537   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2538 }
2539 {
2540   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2541   \bool_lazy_or:nnT
2542     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2543     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2544   {
2545     \@@_error:n { bad-value-for-baseline }
2546     \int_set:Nn \l_tmpa_int 1
2547   }
2548   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2549 }
2550 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2551 \endpgfpicture
2552 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2553 \int_compare:nNnT \l_@@_first_row_int = 0
2554 {
2555   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2556   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2557 }
2558 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2559 }

```

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.

```

2560 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2561 {

```

We will compute the real width of both delimiters used.

```

2562   \dim_zero_new:N \l_@@_real_left_delim_dim
2563   \dim_zero_new:N \l_@@_real_right_delim_dim
2564   \hbox_set:Nn \l_tmpb_box
2565   {
2566     \c_math_toggle_token
2567     \left #1
2568     \vcenter
2569     {
2570       \vbox_to_ht:nn

```

Here, you should use `\box_ht_plus_dp:N` when TeXLive 2021 will be available on Overleaf.

```

2571       { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2572       { }
2573     }
2574     \right .
2575     \c_math_toggle_token
2576   }
2577   \dim_set:Nn \l_@@_real_left_delim_dim

```

```

2578     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2579     \hbox_set:Nn \l_tmpb_box
2580     {
2581         \c_math_toggle_token
2582         \left .
2583         \vbox_to_ht:nn

```

Here, you should use `\box_ht_plus_dp:N` when TeXLive 2021 will be available on Overleaf.

```

2584         { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2585         { }
2586         \right #2
2587         \c_math_toggle_token
2588     }
2589     \dim_set:Nn \l_@@_real_right_delim_dim
2590     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2591     \skip_horizontal:N \l_@@_left_delim_dim
2592     \skip_horizontal:N -\l_@@_real_left_delim_dim
2593     \@@_put_box_in_flow:
2594     \skip_horizontal:N \l_@@_right_delim_dim
2595     \skip_horizontal:N -\l_@@_real_right_delim_dim
2596 }

```

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).

```

2597 \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).

```

2598 {
2599     \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`).

```

2600     { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2601 }
2602 {
2603     \@@_create_col_nodes:
2604     \endarray
2605 }

```

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`).

```

2606 \NewDocumentEnvironment { @@-light-syntax } { b }
2607 {

```

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`.

```

2608     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2609     \tl_map_inline:nn { #1 }
2610     {
2611         \str_if_eq:nnT { ##1 } { & }
2612         { \@@_fatal:n { ampersand-in-light-syntax } }
2613         \str_if_eq:nnT { ##1 } { \ }
2614         { \@@_fatal:n { double-backslash-in-light-syntax } }
2615     }

```


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`.

```
2616   \@@_light_syntax_i #1 \CodeAfter \q_stop
2617 }
```

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`.

```
2618 { }
2619 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2620 {
2621   \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).

```
2622   \seq_gclear_new:N \g_@@_rows_seq
2623   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2624   \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.

```
2625   \int_compare:nNnT \l_@@_last_row_int = { -1 }
2626     { \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`).

```
2627   \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.

```
2628   \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2629   \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
2630   \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2631   \@@_create_col_nodes:
2632   \endarray
2633 }
2634 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2635 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
2636 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2637 {
2638   \seq_gclear_new:N \g_@@_cells_seq
2639   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2640   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2641   \l_tmpa_tl
2642   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2643 }
```

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).

```
2644 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2645 {
2646   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2647     { \@@_fatal:n { empty-environment } } }
```

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

```
2648   \end { #2 }
2649 }
```

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).

```

2650 \cs_new:Npn \@@_create_col_nodes:
2651 {
2652   \crrc
2653   \int_compare:nNnT \l_@@_first_col_int = 0
2654   {
2655     \omit
2656     \hbox_overlap_left:n
2657     {
2658       \bool_if:NT \l_@@_code_before_bool
2659       { \pgfsys@markposition { \@@_env: - col - 0 } }
2660       \pgfpicture
2661       \pgfrememberpicturerepositiononpagetrue
2662       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
2663       \str_if_empty:NF \l_@@_name_str
2664       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2665       \endpgfpicture
2666       \skip_horizontal:N 2\col@sep
2667       \skip_horizontal:N \g_@@_width_first_col_dim
2668     }
2669     &
2670   }
2671   \omit

```

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

```

2672   \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`).

```

2673   \int_compare:nNnTF \l_@@_first_col_int = 0
2674   {
2675     \bool_if:NT \l_@@_code_before_bool
2676     {
2677       \hbox
2678       {
2679         \skip_horizontal:N -0.5\arrayrulewidth
2680         \pgfsys@markposition { \@@_env: - col - 1 }
2681         \skip_horizontal:N 0.5\arrayrulewidth
2682       }
2683     }
2684     \pgfpicture
2685     \pgfrememberpicturerepositiononpagetrue
2686     \pgfcoordinate { \@@_env: - col - 1 }
2687     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
2688     \str_if_empty:NF \l_@@_name_str
2689     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2690     \endpgfpicture
2691   }
2692   {
2693     \bool_if:NT \l_@@_code_before_bool
2694     {
2695       \hbox
2696       {
2697         \skip_horizontal:N 0.5\arrayrulewidth
2698         \pgfsys@markposition { \@@_env: - col - 1 }
2699         \skip_horizontal:N -0.5\arrayrulewidth
2700       }
2701     }
2702     \pgfpicture
2703     \pgfrememberpicturerepositiononpagetrue
2704     \pgfcoordinate { \@@_env: - col - 1 }
2705     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }

```

```

2706     \str_if_empty:NF \l_@@_name_str
2707     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2708     \endpgfpicture
2709 }

```

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.

```

2710     \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2711     \bool_if:NF \l_@@_auto_columns_width_bool
2712     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2713     {
2714         \bool_lazy_and:nnTF
2715         \l_@@_auto_columns_width_bool
2716         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2717         { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2718         { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2719         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2720     }
2721     \skip_horizontal:N \g_tmpa_skip
2722     \hbox
2723     {
2724         \bool_if:NT \l_@@_code_before_bool
2725         {
2726             \hbox
2727             {
2728                 \skip_horizontal:N -0.5\arrayrulewidth
2729                 \pgfsys@markposition { \@@_env: - col - 2 }
2730                 \skip_horizontal:N 0.5\arrayrulewidth
2731             }
2732         }
2733         \pgfpicture
2734         \pgfrememberpicturepositiononpagetrue
2735         \pgfcoordinate { \@@_env: - col - 2 }
2736         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2737         \str_if_empty:NF \l_@@_name_str
2738         { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2739         \endpgfpicture
2740     }

```

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.

```

2741     \int_gset:Nn \g_tmpa_int 1
2742     \bool_if:NTF \g_@@_last_col_found_bool
2743     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2744     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2745     {
2746         &
2747         \omit
2748         \int_gincr:N \g_tmpa_int

```

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

```

2749         \skip_horizontal:N \g_tmpa_skip
2750         \bool_if:NT \l_@@_code_before_bool
2751         {
2752             \hbox
2753             {
2754                 \skip_horizontal:N -0.5\arrayrulewidth
2755                 \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2756                 \skip_horizontal:N 0.5\arrayrulewidth
2757             }

```

2758 }

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

```

2759 \pgfpicture
2760 \pgfrememberpicturepositiononpagetrue
2761 \pgfcoordinate { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
2762 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2763 \str_if_empty:NF \l_@@_name_str
2764 {
2765     \pgfnodealias
2766     { \l_@@_name_str - col - \l_@@_succ:n \g_tmpa_int }
2767     { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
2768 }
2769 \endpgfpicture
2770 }

```

```

2771 &
2772 \omit
2773 \int_gincr:N \g_tmpa_int
2774 \skip_horizontal:N \g_tmpa_skip
2775 \bool_lazy_all:nT
2776 {
2777     \l_@@_NiceArray_bool
2778     { \bool_not_p:n \l_@@_NiceTabular_bool }
2779     { \clist_if_empty_p:N \l_@@_vlines_clist }
2780     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2781     { ! \l_@@_bar_at_end_of_pream_bool }
2782 }
2783 { \skip_horizontal:N -\col@sep }
2784 \bool_if:NT \l_@@_code_before_bool
2785 {
2786     \hbox
2787     {
2788         \skip_horizontal:N -0.5\arrayrulewidth

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

2789     \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2790     { \skip_horizontal:N -\arraycolsep }
2791     \pgfsys@markposition { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
2792     \skip_horizontal:N 0.5\arrayrulewidth
2793     \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2794     { \skip_horizontal:N \arraycolsep }
2795 }
2796 }
2797 \pgfpicture
2798 \pgfrememberpicturepositiononpagetrue
2799 \pgfcoordinate { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }
2800 {
2801     \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
2802     {
2803         \pgfpoint
2804         { - 0.5 \arrayrulewidth - \arraycolsep }
2805         \c_zero_dim
2806     }
2807     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2808 }
2809 \str_if_empty:NF \l_@@_name_str
2810 {
2811     \pgfnodealias
2812     { \l_@@_name_str - col - \l_@@_succ:n \g_tmpa_int }
2813     { \l_@@_env: - col - \l_@@_succ:n \g_tmpa_int }

```

```

2814     }
2815     \endpgfpicture

2816     \bool_if:NT \g_@@_last_col_found_bool
2817     {
2818         \hbox_overlap_right:n
2819         {
2820             \skip_horizontal:N \g_@@_width_last_col_dim
2821             \bool_if:NT \l_@@_code_before_bool
2822             {
2823                 \pgfsys@markposition
2824                 { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2825             }
2826             \pgfpicture
2827             \pgfrememberpicturepositiononpagetrue
2828             \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2829             \pgfpointorigin
2830             \str_if_empty:NF \l_@@_name_str
2831             {
2832                 \pgfnodealias
2833                 { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2834                 { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2835             }
2836             \endpgfpicture
2837         }
2838     }
2839     \cr
2840 }

```

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

```

2841 \tl_const:Nn \c_@@_preamble_first_col_tl
2842 {
2843     >
2844     {

```

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

```

2845     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
2846     \bool_gset_true:N \g_@@_after_col_zero_bool
2847     \@@_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.

```

2848     \hbox_set:Nw \l_@@_cell_box
2849     \@@_math_toggle_token:
2850     \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”.

```

2851     \bool_lazy_and:nnT
2852     { \int_compare_p:nNn \c@iRow > 0 }
2853     {
2854         \bool_lazy_or_p:nn
2855         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2856         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2857     }
2858     {
2859         \l_@@_code_for_first_col_tl
2860         \xglobal \colorlet { nicematrix-first-col } { . }
2861     }
2862 }

```

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`.

```

2863   l
2864   <
2865   {
2866     \@@_math_toggle_token:
2867     \hbox_set_end:
2868     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2869     \@@_adjust_size_box:
2870     \@@_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.

```

2871     \dim_gset:Nn \g_@@_width_first_col_dim
2872     { \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.

```

2873     \hbox_overlap_left:n
2874     {
2875       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2876       \@@_node_for_cell:
2877       { \box_use_drop:N \l_@@_cell_box }
2878       \skip_horizontal:N \l_@@_left_delim_dim
2879       \skip_horizontal:N \l_@@_left_margin_dim
2880       \skip_horizontal:N \l_@@_extra_left_margin_dim
2881     }
2882     \bool_gset_false:N \g_@@_empty_cell_bool
2883     \skip_horizontal:N -2\col@sep
2884   }
2885 }

```

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

```

2886 \tl_const:Nn \c_@@_preamble_last_col_tl
2887 {
2888   >
2889   {

```

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

```

2890     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

2891     \bool_gset_true:N \g_@@_last_col_found_bool
2892     \int_gincr:N \c@jCol
2893     \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.

```

2894     \hbox_set:Nw \l_@@_cell_box
2895     \@@_math_toggle_token:
2896     \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”.

```

2897     \int_compare:nNnT \c@iRow > 0
2898     {
2899       \bool_lazy_or:nnT
2900       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2901       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2902       {
2903         \l_@@_code_for_last_col_tl
2904         \xglobal \colorlet { nicematrix-last-col } { . }
2905       }
2906     }
2907   }
2908   l

```

```

2909 <
2910 {
2911   \@@_math_toggle_token:
2912   \hbox_set_end:
2913   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2914   \@@_adjust_size_box:
2915   \@@_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.

```

2916   \dim_gset:Nn \g_@@_width_last_col_dim
2917   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2918   \skip_horizontal:N -2\col@sep

```

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

```

2919   \hbox_overlap_right:n
2920   {
2921     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2922     {
2923       \skip_horizontal:N \l_@@_right_delim_dim
2924       \skip_horizontal:N \l_@@_right_margin_dim
2925       \skip_horizontal:N \l_@@_extra_right_margin_dim
2926       \@@_node_for_cell:
2927     }
2928   }
2929   \bool_gset_false:N \g_@@_empty_cell_bool
2930 }
2931 }

```

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.

```

2932 \NewDocumentEnvironment { NiceArray } { }
2933 {
2934   \bool_set_true:N \l_@@_NiceArray_bool
2935   \str_if_empty:NT \g_@@_name_env_str
2936   { \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).

```

2937   \NiceArrayWithDelims . .
2938 }
2939 { \endNiceArrayWithDelims }

```

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

```

2940 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2941 {
2942   \NewDocumentEnvironment { #1 NiceArray } { }
2943   {
2944     \str_if_empty:NT \g_@@_name_env_str
2945     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2946     \@@_test_if_math_mode:
2947     \NiceArrayWithDelims #2 #3
2948   }
2949   { \endNiceArrayWithDelims }
2950 }
2951 \@@_def_env:nnn p ( )
2952 \@@_def_env:nnn b [ ]
2953 \@@_def_env:nnn B \{ \}
2954 \@@_def_env:nnn v | |
2955 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2956 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2957 {
2958   \bool_set_true:N \l_@@_Matrix_bool
2959   \use:c { #1 NiceArray }
2960   {
2961     *
2962     {
2963       \int_compare:nNnTF \l_@@_last_col_int < 0
2964       \c@MaxMatrixCols
2965       { \@@_pred:n \l_@@_last_col_int }
2966     }
2967     { > \@@_cell_begin:w #2 < \@@_cell_end: }
2968   }
2969 }
2970 \clist_map_inline:nn { { } , p , b , B , v , V }
2971 {
2972   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2973   {
2974     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2975     \tl_set:Nn \l_@@_type_of_col_tl c
2976     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2977     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2978   }
2979   { \use:c { end #1 NiceArray } }
2980 }

```

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

```

2981 \cs_new_protected:Npn \@@_NotEmpty:
2982 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

{NiceTabular}, {NiceTabularX} and {NiceTabular*}

```

2983 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2984 {

```

If the dimension \l_@@_width_dim is equal to 0 pt, that means that it has not be set by a previous use of \NiceMatrixOptions.

```

2985   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
2986   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
2987   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2988   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2989   \bool_set_true:N \l_@@_NiceTabular_bool
2990   \NiceArray { #2 }
2991 }
2992 { \endNiceArray }

2993 \cs_set_protected:Npn \@@_newcolumnntype #1
2994 {
2995   \cs_if_free:cT { NC @ find @ #1 }
2996   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
2997   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
2998   \peek_meaning:NTF [
2999     { \newcol@ #1 }
3000     { \newcol@ #1 [ 0 ] }
3001   }

```

```

3002 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3003 {

```

The following code prevents the expansion of the ‘X’ columns with the definition of that columns in tabularx (this would result in an error in {NiceTabularX}).


```

3004 \bool_if:NT \c_@@_tabularx_loaded_bool
3005 { \newcolumnntype { X } { \@@_X } }
3006 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3007 \dim_zero_new:N \l_@@_width_dim
3008 \dim_set:Nn \l_@@_width_dim { #1 }
3009 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3010 \bool_set_true:N \l_@@_NiceTabular_bool
3011 \NiceArray { #3 }
3012 }
3013 { \endNiceArray }

3014 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3015 {
3016 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3017 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3018 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3019 \bool_set_true:N \l_@@_NiceTabular_bool
3020 \NiceArray { #3 }
3021 }
3022 { \endNiceArray }

```

After the construction of the array

```

3023 \cs_new_protected:Npn \@@_after_array:
3024 {
3025 \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.

```

3026 \bool_if:NT \g_@@_last_col_found_bool
3027 { \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 also fix the real value of `\l_@@_last_col_int`.

```

3028 \bool_if:NT \l_@@_last_col_without_value_bool
3029 { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }

```

It's also time to give to `\l_@@_last_row_int` its real value.

```

3030 \bool_if:NT \l_@@_last_row_without_value_bool
3031 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3032 \tl_gput_right:Nx \g_@@_aux_tl
3033 {
3034 \seq_gset_from_clist:Nn \exp_not:N \c_@@_size_seq
3035 {
3036 \int_use:N \l_@@_first_row_int ,
3037 \int_use:N \c_iRow ,
3038 \int_use:N \g_@@_row_total_int ,
3039 \int_use:N \l_@@_first_col_int ,
3040 \int_use:N \c_jCol ,
3041 \int_use:N \g_@@_col_total_int
3042 }
3043 }

```

We write also the potential content of `\g_@@_pos_of_blocks_seq`. It will be used to recreate the blocks with a name in the `\CodeBefore` and also if the command `\rowcolors` is used with the key `respect-blocks`).

```

3044 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3045 {

```

```

3046     \tl_gput_right:Nx \g_@@_aux_tl
3047     {
3048         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3049         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3050     }
3051 }
3052 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3053 {
3054     \tl_gput_right:Nx \g_@@_aux_tl
3055     {
3056         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3057         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3058         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3059         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3060     }
3061 }

```

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

```

3062     \@@_create_diag_nodes:

```

We create the aliases using `last` for the nodes of the cells in the last row and the last column.

```

3063     \pgfpicture
3064     \int_step_inline:nn \c@iRow
3065     {
3066         \pgfnodealias
3067         { \@@_env: - ##1 - last }
3068         { \@@_env: - ##1 - \int_use:N \c@jCol }
3069     }
3070     \int_step_inline:nn \c@jCol
3071     {
3072         \pgfnodealias
3073         { \@@_env: - last - ##1 }
3074         { \@@_env: - \int_use:N \c@iRow - ##1 }
3075     }
3076     \str_if_empty:NF \l_@@_name_str
3077     {
3078         \int_step_inline:nn \c@iRow
3079         {
3080             \pgfnodealias
3081             { \l_@@_name_str - ##1 - last }
3082             { \@@_env: - ##1 - \int_use:N \c@jCol }
3083         }
3084         \int_step_inline:nn \c@jCol
3085         {
3086             \pgfnodealias
3087             { \l_@@_name_str - last - ##1 }
3088             { \@@_env: - \int_use:N \c@iRow - ##1 }
3089         }
3090     }
3091     \endpgfpicture

```

By default, the diagonal lines will be parallelized⁶⁹. 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.

```

3092     \bool_if:NT \l_@@_parallelize_diags_bool
3093     {
3094         \int_gzero_new:N \g_@@_ddots_int
3095         \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`

⁶⁹It's possible to use the option `parallelize-diags` to disable this parallelization.

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.

```

3096     \dim_gzero_new:N \g_@@_delta_x_one_dim
3097     \dim_gzero_new:N \g_@@_delta_y_one_dim
3098     \dim_gzero_new:N \g_@@_delta_x_two_dim
3099     \dim_gzero_new:N \g_@@_delta_y_two_dim
3100   }
3101   \int_zero_new:N \l_@@_initial_i_int
3102   \int_zero_new:N \l_@@_initial_j_int
3103   \int_zero_new:N \l_@@_final_i_int
3104   \int_zero_new:N \l_@@_final_j_int
3105   \bool_set_false:N \l_@@_initial_open_bool
3106   \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 `\vdottedline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.

```

3107   \bool_if:NT \l_@@_small_bool
3108   {
3109     \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
3110     \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.

```

3111     \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
3112   }

```

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

```

3113   \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won't do anything. The corners are computed in `\l_@@_corners_cells_seq` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

3114   \@@_compute_corners:

```

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

```

3115   \@@_adjust_pos_of_blocks_seq:
3116   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3117   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
3118   \cs_set_eq:NN \SubMatrix \@@_SubMatrix

```

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

```

3119   \bool_if:NT \c_@@_tikz_loaded_bool
3120   {
3121     \tikzset
3122     {
3123       every~picture / .style =
3124       {
3125         overlay ,
3126         remember~picture ,
3127         name~prefix = \@@_env: -
3128       }
3129     }
3130   }
3131   \cs_set_eq:NN \line \@@_line
3132   \g_@@_internal_code_after_tl
3133   \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.

```
3134 \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).

```
3135 \seq_gclear:N \g_@@_submatrix_names_seq
```

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

```
3136 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
```

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:`.

```
3137 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
```

```
3138 \scan_stop:
```

```
3139 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
```

```
3140 \tl_gclear:N \g_nicematrix_code_after_tl
```

```
3141 \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.

```
3142 \tl_if_empty:N \g_nicematrix_code_before_tl
```

```
3143 {
```

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`.

```
3144 \cs_set_protected:Npn \rectanglecolor { }
```

```
3145 \cs_set_protected:Npn \columncolor { }
```

```
3146 \tl_gput_right:Nx \g_@@_aux_tl
```

```
3147 {
```

```
3148 \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
```

```
3149 { \exp_not:N \g_nicematrix_code_before_tl }
```

```
3150 }
```

```
3151 \bool_set_true:N \l_@@_code_before_bool
```

```
3152 }
```

```
3153 \str_gclear:N \g_@@_name_env_str
```

```
3154 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁷⁰. 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`.

```
3155 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
```

```
3156 }
```

The following command will extract the potential options (between square brackets) at the beginning of the `\CodeAfter` (that is to say, when `\CodeAfter` is used, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```
3157 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
```

```
3158 { \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

⁷⁰e.g. `\color[rgb]{0.5,0.5,0}`

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).

```

3159 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3160 {
3161     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3162     { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3163 }

```

The following command must *not* be protected.

```

3164 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3165 {
3166     { #1 }
3167     { #2 }
3168     {
3169         \int_compare:nNnTF { #3 } > { 99 }
3170         { \int_use:N \c@iRow }
3171         { #3 }
3172     }
3173     {
3174         \int_compare:nNnTF { #4 } > { 99 }
3175         { \int_use:N \c@jCol }
3176         { #4 }
3177     }
3178     { #5 }
3179 }

```

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).

```

3180 \AtBeginDocument
3181 {
3182     \cs_new_protected:Npx \@@_draw_dotted_lines:
3183     {
3184         \c_@@_pgfortikzpicture_tl
3185         \@@_draw_dotted_lines_i:
3186         \c_@@_endpgfortikzpicture_tl
3187     }
3188 }

```

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

```

3189 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3190 {
3191     \pgfrememberpicturerepositiononpagetrue
3192     \pgf@relevantforpicturesizefalse
3193     \g_@@_HVdotsfor_lines_tl
3194     \g_@@_Vdots_lines_tl
3195     \g_@@_Ddots_lines_tl
3196     \g_@@_Iddots_lines_tl
3197     \g_@@_Cdots_lines_tl
3198     \g_@@_Ldots_lines_tl
3199 }

3200 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3201 {
3202     \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3203     \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3204 }

```

We define a new PGF shape for the diag nodes because we want to provide a anchor called `.5` for those nodes.

```

3205 \pgfdeclareshape { @@_diag_node }
3206 {

```

```

3207 \savedanchor { \five }
3208 {
3209     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3210     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3211 }
3212 \anchor { 5 } { \five }
3213 \anchor { center } { \pgfpointorigin }
3214 }

```

The following command creates the diagonal nodes (in fact, if the matrix is not a square matrix, not all the nodes are on the diagonal).

```

3215 \cs_new_protected:Npn \@@_create_diag_nodes:
3216 {
3217     \pgfpicture
3218     \pgfrememberpicturepositiononpagetrue
3219     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3220     {
3221         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3222         \dim_set_eq:NN \l_tmpa_dim \pgf@x
3223         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3224         \dim_set_eq:NN \l_tmpb_dim \pgf@y
3225         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3226         \dim_set_eq:NN \l_tmpc_dim \pgf@x
3227         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3228         \dim_set_eq:NN \l_tmpd_dim \pgf@y
3229         \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@â_diag_node`) that we will construct.

```

3230     \dim_set:Nn \l_tmpa_dim { ( \l_tmpc_dim - \l_tmpa_dim ) / 2 }
3231     \dim_set:Nn \l_tmpb_dim { ( \l_tmpd_dim - \l_tmpb_dim ) / 2 }
3232     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3233     \str_if_empty:NF \l_@@_name_str
3234     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3235 }

```

Now, the last node. Of course, that is only a coordinate because there is not `.5` anchor for that node.

```

3236 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3237 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3238 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3239 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3240 \pgfcoordinate
3241 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3242 \pgfnodealias
3243 { \@@_env: - last }
3244 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3245 \str_if_empty:NF \l_@@_name_str
3246 {
3247     \pgfnodealias
3248     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3249     { \@@_env: - \int_use:N \l_tmpa_int }
3250     \pgfnodealias
3251     { \l_@@_name_str - last }
3252     { \@@_env: - last }
3253 }
3254 \endpgfpicture
3255 }

```

We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on

its right.

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

The command `\l_@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
3256 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3257 {
```

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

```
3258 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3259 \int_set:Nn \l_@@_initial_i_int { #1 }
3260 \int_set:Nn \l_@@_initial_j_int { #2 }
3261 \int_set:Nn \l_@@_final_i_int { #1 }
3262 \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.

```
3263 \bool_set_false:N \l_@@_stop_loop_bool
3264 \bool_do_until:Nn \l_@@_stop_loop_bool
3265 {
3266   \int_add:Nn \l_@@_final_i_int { #3 }
3267   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3268 \bool_set_false:N \l_@@_final_open_bool
3269 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3270 {
3271   \int_compare:nNnTF { #3 } = 1
3272   { \bool_set_true:N \l_@@_final_open_bool }
3273   {
3274     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3275     { \bool_set_true:N \l_@@_final_open_bool }
3276   }
3277 }
3278 {
3279   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3280   {
3281     \int_compare:nNnTF { #4 } = { -1 }
3282     { \bool_set_true:N \l_@@_final_open_bool }
3283   }
3284 }
```

```

3285         \int_compare:nNtT \l_@@_final_j_int > \l_@@_col_max_int
3286         {
3287             \int_compare:nNtT { #4 } = 1
3288             { \bool_set_true:N \l_@@_final_open_bool }
3289         }
3290     }
3291 }
3292 \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.

```

3293 {

```

We do a step backwards.

```

3294     \int_sub:Nn \l_@@_final_i_int { #3 }
3295     \int_sub:Nn \l_@@_final_j_int { #4 }
3296     \bool_set_true:N \l_@@_stop_loop_bool
3297 }

```

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`.

```

3298 {
3299     \cs_if_exist:cTF
3300     {
3301         @@ _ dotted _
3302         \int_use:N \l_@@_final_i_int -
3303         \int_use:N \l_@@_final_j_int
3304     }
3305     {
3306         \int_sub:Nn \l_@@_final_i_int { #3 }
3307         \int_sub:Nn \l_@@_final_j_int { #4 }
3308         \bool_set_true:N \l_@@_final_open_bool
3309         \bool_set_true:N \l_@@_stop_loop_bool
3310     }
3311     {
3312         \cs_if_exist:cTF
3313         {
3314             pgf @ sh @ ns @ \@@_env:
3315             - \int_use:N \l_@@_final_i_int
3316             - \int_use:N \l_@@_final_j_int
3317         }
3318         { \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.

```

3319     {
3320         \cs_set:cpn
3321         {
3322             @@ _ dotted _
3323             \int_use:N \l_@@_final_i_int -
3324             \int_use:N \l_@@_final_j_int
3325         }
3326         { }
3327     }
3328 }
3329 }
3330 }

```

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

```

3331     \bool_set_false:N \l_@@_stop_loop_bool

```



```

3332 \bool_do_until:Nn \l_@@_stop_loop_bool
3333 {
3334   \int_sub:Nn \l_@@_initial_i_int { #3 }
3335   \int_sub:Nn \l_@@_initial_j_int { #4 }
3336   \bool_set_false:N \l_@@_initial_open_bool
3337   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3338   {
3339     \int_compare:nNnTF { #3 } = 1
3340     { \bool_set_true:N \l_@@_initial_open_bool }
3341     {
3342       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3343       { \bool_set_true:N \l_@@_initial_open_bool }
3344     }
3345   }
3346   {
3347     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3348     {
3349       \int_compare:nNnT { #4 } = 1
3350       { \bool_set_true:N \l_@@_initial_open_bool }
3351     }
3352     {
3353       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3354       {
3355         \int_compare:nNnT { #4 } = { -1 }
3356         { \bool_set_true:N \l_@@_initial_open_bool }
3357       }
3358     }
3359   }
3360   \bool_if:NnTF \l_@@_initial_open_bool
3361   {
3362     \int_add:Nn \l_@@_initial_i_int { #3 }
3363     \int_add:Nn \l_@@_initial_j_int { #4 }
3364     \bool_set_true:N \l_@@_stop_loop_bool
3365   }
3366   {
3367     \cs_if_exist:cTF
3368     {
3369       @@ _ dotted _
3370       \int_use:N \l_@@_initial_i_int -
3371       \int_use:N \l_@@_initial_j_int
3372     }
3373     {
3374       \int_add:Nn \l_@@_initial_i_int { #3 }
3375       \int_add:Nn \l_@@_initial_j_int { #4 }
3376       \bool_set_true:N \l_@@_initial_open_bool
3377       \bool_set_true:N \l_@@_stop_loop_bool
3378     }
3379     {
3380       \cs_if_exist:cTF
3381       {
3382         pgf @ sh @ ns @ \@@_env:
3383         - \int_use:N \l_@@_initial_i_int
3384         - \int_use:N \l_@@_initial_j_int
3385       }
3386       { \bool_set_true:N \l_@@_stop_loop_bool }
3387       {
3388         \cs_set:cpn
3389         {
3390           @@ _ dotted _
3391           \int_use:N \l_@@_initial_i_int -
3392           \int_use:N \l_@@_initial_j_int
3393         }
3394         { }

```

```

3395         }
3396     }
3397 }
3398 }

```

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.

```

3399 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3400 {
3401     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

3402     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3403     { \int_use:N \l_@@_final_i_int }
3404     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3405     { } % for the name of the block
3406 }
3407 }

```

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 rows and columns).

```

3408 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3409 {
3410     \int_set:Nn \l_@@_row_min_int 1
3411     \int_set:Nn \l_@@_col_min_int 1
3412     \int_set_eq:NN \l_@@_row_max_int \c{iRow}
3413     \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`.

```

3414 \seq_map_inline:Nn \g_@@_submatrix_seq
3415 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
3416 }

```

#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.

```

3417 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3418 {
3419     \bool_if:nT
3420     {
3421         \int_compare_p:n { #3 <= #1 }
3422         && \int_compare_p:n { #1 <= #5 }
3423         && \int_compare_p:n { #4 <= #2 }
3424         && \int_compare_p:n { #2 <= #6 }
3425     }
3426     {
3427         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3428         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3429         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3430         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3431     }
3432 }

```

```

3433 \cs_new_protected:Npn \@@_set_initial_coords:
3434 {
3435     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3436     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3437 }
3438 \cs_new_protected:Npn \@@_set_final_coords:
3439 {

```

```

3440 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3441 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3442 }
3443 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3444 {
3445   \pgfpointanchor
3446   {
3447     \@@_env:
3448     - \int_use:N \l_@@_initial_i_int
3449     - \int_use:N \l_@@_initial_j_int
3450   }
3451   { #1 }
3452   \@@_set_initial_coords:
3453 }
3454 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3455 {
3456   \pgfpointanchor
3457   {
3458     \@@_env:
3459     - \int_use:N \l_@@_final_i_int
3460     - \int_use:N \l_@@_final_j_int
3461   }
3462   { #1 }
3463   \@@_set_final_coords:
3464 }
3465 \cs_new_protected:Npn \@@_open_x_initial_dim:
3466 {
3467   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3468   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3469   {
3470     \cs_if_exist:cT
3471     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3472     {
3473       \pgfpointanchor
3474       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3475       { west }
3476       \dim_set:Nn \l_@@_x_initial_dim
3477       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3478     }
3479   }

```

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

```

3480   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3481   {
3482     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3483     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3484     \dim_add:Nn \l_@@_x_initial_dim \col@sep
3485   }
3486 }
3487 \cs_new_protected:Npn \@@_open_x_final_dim:
3488 {
3489   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3490   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3491   {
3492     \cs_if_exist:cT
3493     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3494     {
3495       \pgfpointanchor
3496       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3497       { east }
3498       \dim_set:Nn \l_@@_x_final_dim
3499       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3500     }

```

```
3501 }
```

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

```
3502 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3503 {
3504   \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3505   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3506   \dim_sub:Nn \l_@@_x_final_dim \col@sep
3507 }
3508 }
```

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.

```
3509 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3510 {
3511   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3512   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3513   {
3514     \@@_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.

```
3515   \group_begin:
3516   \int_compare:nNnTF { #1 } = 0
3517     { \color { nicematrix-first-row } }
3518   {
```

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.

```
3519     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3520     { \color { nicematrix-last-row } }
3521   }
3522   \keys_set:nn { NiceMatrix / xdots } { #3 }
3523   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3524   \@@_actually_draw_Ldots:
3525   \group_end:
3526 }
3527 }
```

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`.

```
3528 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3529 {
3530   \bool_if:NTF \l_@@_initial_open_bool
3531   {
3532     \@@_open_x_initial_dim:
3533     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3534     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3535   }
3536   { \@@_set_initial_coords_from_anchor:n { base-east } }
3537   \bool_if:NTF \l_@@_final_open_bool
```

```

3538 {
3539   \@@_open_x_final_dim:
3540   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3541   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3542 }
3543 { \@@_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 (?).

```

3544   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3545   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3546   \@@_draw_line:
3547 }

```

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.

```

3548 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3549 {
3550   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3551   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3552   {
3553     \@@_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.

```

3554     \group_begin:
3555     \int_compare:nNnTF { #1 } = 0
3556     { \color { nicematrix-first-row } }
3557     {

```

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.

```

3558       \int_compare:nNnT { #1 } = \l_@@_last_row_int
3559       { \color { nicematrix-last-row } }
3560     }
3561     \keys_set:nn { NiceMatrix / xdots } { #3 }
3562     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3563     \@@_actually_draw_Cdots:
3564   \group_end:
3565 }
3566 }

```

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`.

```

3567 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3568 {
3569   \bool_if:NTF \l_@@_initial_open_bool
3570   { \@@_open_x_initial_dim: }
3571   { \@@_set_initial_coords_from_anchor:n { mid~east } }
3572   \bool_if:NTF \l_@@_final_open_bool
3573   { \@@_open_x_final_dim: }
3574   { \@@_set_final_coords_from_anchor:n { mid~west } }
3575   \bool_lazy_and:nnTF

```

```

3576 \l_@@_initial_open_bool
3577 \l_@@_final_open_bool
3578 {
3579   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3580   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3581   \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
3582   \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3583   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3584 }
3585 {
3586   \bool_if:NT \l_@@_initial_open_bool
3587   { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3588   \bool_if:NT \l_@@_final_open_bool
3589   { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3590 }
3591 \@@_draw_line:
3592 }
3593 \cs_new_protected:Npn \@@_open_y_initial_dim:
3594 {
3595   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3596   \dim_set:Nn \l_@@_y_initial_dim
3597   { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3598   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3599   {
3600     \cs_if_exist:cT
3601     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3602     {
3603       \pgfpointanchor
3604       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3605       { north }
3606       \dim_set:Nn \l_@@_y_initial_dim
3607       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3608     }
3609   }
3610 }
3611 \cs_new_protected:Npn \@@_open_y_final_dim:
3612 {
3613   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3614   \dim_set:Nn \l_@@_y_final_dim
3615   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3616   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3617   {
3618     \cs_if_exist:cT
3619     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3620     {
3621       \pgfpointanchor
3622       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3623       { south }
3624       \dim_set:Nn \l_@@_y_final_dim
3625       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3626     }
3627   }
3628 }

```

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.

```

3629 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3630 {
3631   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3632   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3633   {
3634     \@@_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.

```

3635     \group_begin:
3636     \int_compare:nNnTF { #2 } = 0
3637     { \color { nicematrix-first-col } }
3638     {
3639         \int_compare:nNnT { #2 } = \l_@@_last_col_int
3640         { \color { nicematrix-last-col } }
3641     }
3642     \keys_set:nn { NiceMatrix / xdots } { #3 }
3643     \tl_if_empty:VF \l_@@_xdots_color_tl
3644     { \color { \l_@@_xdots_color_tl } }
3645     \@@_actually_draw_Vdots:
3646 \group_end:
3647 }
3648 }
```

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`.

```

3649 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3650 {
```

The boolean `\l_tmpa_bool` indicates whether the column is of type 1 or may be considered as if.

```

3651     \bool_set_false:N \l_tmpa_bool
```

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

```

3652     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3653     {
3654         \@@_set_initial_coords_from_anchor:n { south-west }
3655         \@@_set_final_coords_from_anchor:n { north-west }
3656         \bool_set:Nn \l_tmpa_bool
3657         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3658     }
```

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

```

3659     \bool_if:NTF \l_@@_initial_open_bool
3660     \@@_open_y_initial_dim:
3661     { \@@_set_initial_coords_from_anchor:n { south } }
3662     \bool_if:NTF \l_@@_final_open_bool
3663     \@@_open_y_final_dim:
3664     { \@@_set_final_coords_from_anchor:n { north } }
3665     \bool_if:NTF \l_@@_initial_open_bool
3666     {
3667         \bool_if:NTF \l_@@_final_open_bool
3668         {
3669             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3670             \dim_set_eq:NN \l_tmpa_dim \pgf@x
3671             \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3672             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3673             \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.

```

3674         \int_compare:nNnT \l_@@_last_col_int > { -2 }
3675         {
3676             \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3677             {
3678                 \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3679                 \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3680                 \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3681                 \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3682             }
3683         }
3684     }
3685     { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3686 }
3687 {
3688     \bool_if:NTF \l_@@_final_open_bool
3689     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3690 }

```

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.

```

3691         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3692         {
3693             \dim_set:Nn \l_@@_x_initial_dim
3694             {
3695                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3696                 \l_@@_x_initial_dim \l_@@_x_final_dim
3697             }
3698             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3699         }
3700     }
3701 }
3702 \@@_draw_line:
3703 }

```

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.

```

3704 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3705 {
3706     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3707     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3708     {
3709         \@@_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.

```

3710     \group_begin:
3711     \keys_set:nn { NiceMatrix / xdots } { #3 }
3712     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3713     \@@_actually_draw_Ddots:
3714     \group_end:
3715 }
3716 }

```

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.

```

3717 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3718 {
3719   \bool_if:NTF \l_@@_initial_open_bool
3720   {
3721     \@@_open_y_initial_dim:
3722     \@@_open_x_initial_dim:
3723   }
3724   { \@@_set_initial_coords_from_anchor:n { south-east } }
3725   \bool_if:NTF \l_@@_final_open_bool
3726   {
3727     \@@_open_x_final_dim:
3728     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3729   }
3730   { \@@_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.

```

3731   \bool_if:NT \l_@@_parallelize_diags_bool
3732   {
3733     \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).

```

3734     \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.

```

3735     {
3736       \dim_gset:Nn \g_@@_delta_x_one_dim
3737       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3738       \dim_gset:Nn \g_@@_delta_y_one_dim
3739       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3740     }

```

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.

```

3741     {
3742       \dim_set:Nn \l_@@_y_final_dim
3743       {
3744         \l_@@_y_initial_dim +
3745         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3746         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3747       }
3748     }
3749   }
3750   \@@_draw_line:
3751 }

```

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.

```

3752 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3753 {
3754   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3755   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }

```

```

3756 {
3757   \@@_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.

```

3758   \group_begin:
3759     \keys_set:nn { NiceMatrix / xdots } { #3 }
3760     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3761     \@@_actually_draw_Iddots:
3762   \group_end:
3763 }
3764 }

```

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.`

```

3765 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3766 {
3767   \bool_if:NTF \l_@@_initial_open_bool
3768   {
3769     \@@_open_y_initial_dim:
3770     \@@_open_x_initial_dim:
3771   }
3772   { \@@_set_initial_coords_from_anchor:n { south-west } }
3773   \bool_if:NTF \l_@@_final_open_bool
3774   {
3775     \@@_open_y_final_dim:
3776     \@@_open_x_final_dim:
3777   }
3778   { \@@_set_final_coords_from_anchor:n { north-east } }
3779   \bool_if:NT \l_@@_parallelize_diags_bool
3780   {
3781     \int_gincr:N \g_@@_iddots_int
3782     \int_compare:nNnTF \g_@@_iddots_int = 1
3783     {
3784       \dim_gset:Nn \g_@@_delta_x_two_dim
3785       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3786       \dim_gset:Nn \g_@@_delta_y_two_dim
3787       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3788     }
3789     {
3790       \dim_set:Nn \l_@@_y_final_dim
3791       {
3792         \l_@@_y_initial_dim +
3793         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3794         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3795       }
3796     }
3797   }
3798   \@@_draw_line:
3799 }

```

The actual instructions for drawing the dotted lines 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`

```

3800 \cs_new_protected:Npn \@@_draw_line:
3801 {
3802   \pgfrememberpicturepositiononpagetrue
3803   \pgf@relevantforpicturesizefalse
3804   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
3805     \@@_draw_standard_dotted_line:
3806     \@@_draw_unstandard_dotted_line:
3807 }

```

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.

```

3808 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
3809 {
3810   \begin { scope }
3811   \exp_args:No \@@_draw_unstandard_dotted_line:n
3812     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3813 }

```

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_unstandard_dotted_line:n` is, in fact, the list of options.

```

3814 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
3815 {
3816   \@@_draw_unstandard_dotted_line:nVV
3817   { #1 }
3818   \l_@@_xdots_up_tl
3819   \l_@@_xdots_down_tl
3820 }
3821 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
3822 {
3823   \draw
3824   [
3825     #1 ,
3826     shorten~> = \l_@@_xdots_shorten_dim ,
3827     shorten~< = \l_@@_xdots_shorten_dim ,
3828   ]
3829     ( \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).

```

3830   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3831   node [ sloped , below ] { $ \scriptstyle #3 $ }
3832   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3833 \end { scope }
3834 }
3835 \cs_generate_variant:Nn \@@_draw_unstandard_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).

```

3836 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3837 {
3838   \bool_lazy_and:nnF
3839     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3840     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3841   {
3842     \pgfscope
3843     \pgftransformshift
3844       {
3845         \pgfpointlineattime { 0.5 }
3846         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3847         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3848       }
3849     \pgftransformrotate
3850       {
3851         \fp_eval:n
3852           {
3853             atand
3854             (
3855               \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3856               \l_@@_x_final_dim - \l_@@_x_initial_dim
3857             )
3858           }
3859       }
3860     \pgfnode
3861       { rectangle }
3862       { south }
3863       {
3864         \c_math_toggle_token
3865         \scriptstyle \l_@@_xdots_up_tl
3866         \c_math_toggle_token
3867       }
3868       { }
3869       { \pgfusepath { } }
3870     \pgfnode
3871       { rectangle }
3872       { north }
3873       {
3874         \c_math_toggle_token
3875         \scriptstyle \l_@@_xdots_down_tl
3876         \c_math_toggle_token
3877       }
3878       { }
3879       { \pgfusepath { } }
3880     \endpgfscope
3881   }
3882   \group_begin:

```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

3883   \dim_zero_new:N \l_@@_l_dim
3884   \dim_set:Nn \l_@@_l_dim
3885     {
3886       \fp_to_dim:n
3887         {
3888           sqrt
3889             (
3890               ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3891               +
3892               ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3893             )
3894         }

```

3895 }

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.

```

3896 \bool_lazy_or:nnF
3897 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3898 { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3899 \@@_draw_standard_dotted_line_i:
3900 \group_end:
3901 }
3902 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3903 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3904 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

3905 \bool_if:NTF \l_@@_initial_open_bool
3906 {
3907   \bool_if:NTF \l_@@_final_open_bool
3908   {
3909     \int_set:Nn \l_tmpa_int
3910     { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3911   }
3912   {
3913     \int_set:Nn \l_tmpa_int
3914     {
3915       \dim_ratio:nn
3916       { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3917       \l_@@_inter_dots_dim
3918     }
3919   }
3920 }
3921 {
3922   \bool_if:NTF \l_@@_final_open_bool
3923   {
3924     \int_set:Nn \l_tmpa_int
3925     {
3926       \dim_ratio:nn
3927       { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3928       \l_@@_inter_dots_dim
3929     }
3930   }
3931   {
3932     \int_set:Nn \l_tmpa_int
3933     {
3934       \dim_ratio:nn
3935       { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3936       \l_@@_inter_dots_dim
3937     }
3938   }
3939 }

```

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

```

3940 \dim_set:Nn \l_tmpa_dim
3941 {
3942   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3943   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3944 }
3945 \dim_set:Nn \l_tmpb_dim
3946 {
3947   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *

```

```

3948     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3949 }

```

The length ℓ is the length of the dotted line. We note Δ the length between two dots and n the number of intervals between dots. We note $\delta = \frac{1}{2}(\ell - n\Delta)$. The distance between the initial extremity of the line and the first dot will be equal to $k \cdot \delta$ where $k = 0, 1$ or 2 . We first compute this number k in `\l_tmpb_int`.

```

3950 \int_set:Nn \l_tmpb_int
3951 {
3952   \bool_if:NTF \l_@@_initial_open_bool
3953     { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3954     { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3955 }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

3956 \dim_gadd:Nn \l_@@_x_initial_dim
3957 {
3958   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3959   \dim_ratio:nn
3960   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3961   { 2 \l_@@_l_dim }
3962   * \l_tmpb_int
3963 }
3964 \dim_gadd:Nn \l_@@_y_initial_dim
3965 {
3966   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3967   \dim_ratio:nn
3968   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3969   { 2 \l_@@_l_dim }
3970   * \l_tmpb_int
3971 }
3972 \pgf@relevantforpicturesizefalse
3973 \int_step_inline:nnn 0 \l_tmpa_int
3974 {
3975   \pgfpathcircle
3976   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3977   { \l_@@_radius_dim }
3978   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3979   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3980 }
3981 \pgfusepathqfill
3982 }

```

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 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.

```

3983 \AtBeginDocument
3984 {
3985   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3986   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

3987 \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3988 {
3989   \int_compare:nNnTF \c@jCol = 0
3990   { \@@_error:nn { in~first~col } \Ldots }
3991   {
3992     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3993     { \@@_error:nn { in~last~col } \Ldots }
3994     {
3995       \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
3996       { #1 , down = #2 , up = #3 }
3997     }
3998   }
3999   \bool_if:NF \l_@@_nullify_dots_bool
4000   { \phantom { \ensuremath { \@@_old_ldots } } }
4001   \bool_gset_true:N \g_@@_empty_cell_bool
4002 }

4003 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4004 {
4005   \int_compare:nNnTF \c@jCol = 0
4006   { \@@_error:nn { in~first~col } \Cdots }
4007   {
4008     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4009     { \@@_error:nn { in~last~col } \Cdots }
4010     {
4011       \@@_instruction_of_type:nnn \c_false_bool { \Cdots }
4012       { #1 , down = #2 , up = #3 }
4013     }
4014   }
4015   \bool_if:NF \l_@@_nullify_dots_bool
4016   { \phantom { \ensuremath { \@@_old_cdots } } }
4017   \bool_gset_true:N \g_@@_empty_cell_bool
4018 }

4019 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4020 {
4021   \int_compare:nNnTF \c@iRow = 0
4022   { \@@_error:nn { in~first~row } \Vdots }
4023   {
4024     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4025     { \@@_error:nn { in~last~row } \Vdots }
4026     {
4027       \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4028       { #1 , down = #2 , up = #3 }
4029     }
4030   }
4031   \bool_if:NF \l_@@_nullify_dots_bool
4032   { \phantom { \ensuremath { \@@_old_vdots } } }
4033   \bool_gset_true:N \g_@@_empty_cell_bool
4034 }

4035 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4036 {
4037   \int_case:nnF \c@iRow
4038   {
4039     0 { \@@_error:nn { in~first~row } \Ddots }
4040     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4041   }
4042   {
4043     \int_case:nnF \c@jCol
4044     {

```

```

4045         0 { \@@_error:nn { in~first~col } \Ddots }
4046     \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4047 }
4048 {
4049     \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4050     \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4051     { #1 , down = #2 , up = #3 }
4052 }
4053
4054 }
4055 \bool_if:NF \l_@@_nullify_dots_bool
4056 { \phantom { \ensuremath { \@@_old_ddots } } }
4057 \bool_gset_true:N \g_@@_empty_cell_bool
4058 }

\exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
{
    \int_case:nnF \c@iRow
    {
        0 { \@@_error:nn { in~first~row } \Iddots }
        \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
    }
    {
        \int_case:nnF \c@jCol
        {
            0 { \@@_error:nn { in~first~col } \Iddots }
            \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
        }
        {
            \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
            \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
            { #1 , down = #2 , up = #3 }
        }
    }
    \bool_if:NF \l_@@_nullify_dots_bool
    { \phantom { \ensuremath { \@@_old_iddots } } }
    \bool_gset_true:N \g_@@_empty_cell_bool
}
}

```

End of the \AtBeginDocument.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

4083 \keys_define:nn { NiceMatrix / Ddots }
4084 {
4085     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4086     draw-first .default:n = true ,
4087     draw-first .value_forbidden:n = true
4088 }

```

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

```

4089 \cs_new_protected:Npn \@@_Hspace:
4090 {
4091     \bool_gset_true:N \g_@@_empty_cell_bool
4092     \hspace
4093 }

```

In the environments of nicematrix, the command \multicolumn is redefined. We will patch the environment {tabular} to go back to the previous value of \multicolumn.

```

4094 \cs_set_eq:NN \@@_old_multicolumn \multicolumn

```


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`.

```

4095 \cs_new:Npn \@@_Hdotsfor:
4096 {
4097   \bool_lazy_and:nnTF
4098     { \int_compare_p:nNn \c@jCol = 0 }
4099     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4100     {
4101       \bool_if:NTF \g_@@_after_col_zero_bool
4102       {
4103         \multicolumn { 1 } { c } { }
4104         \@@_Hdotsfor_i
4105       }
4106       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4107     }
4108     {
4109       \multicolumn { 1 } { c } { }
4110       \@@_Hdotsfor_i
4111     }
4112 }

```

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:`).

```

4113 \AtBeginDocument
4114 {
4115   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4116   \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.

```

4117   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4118   {
4119     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4120     {
4121       \@@_Hdotsfor:nnnn
4122       { \int_use:N \c@iRow }
4123       { \int_use:N \c@jCol }
4124       { #2 }
4125       {
4126         #1 , #3 ,
4127         down = \exp_not:n { #4 } ,
4128         up = \exp_not:n { #5 }
4129       }
4130     }
4131     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4132   }
4133 }

```

Enf of `\AtBeginDocument`.

```

4134 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4135 {
4136   \bool_set_false:N \l_@@_initial_open_bool
4137   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4138   \int_set:Nn \l_@@_initial_i_int { #1 }
4139   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4140   \int_compare:nNnTF { #2 } = 1
4141   {

```

```

4142     \int_set:Nn \l_@@_initial_j_int 1
4143     \bool_set_true:N \l_@@_initial_open_bool
4144   }
4145   {
4146     \cs_if_exist:cTF
4147     {
4148       pgf @ sh @ ns @ \@@_env:
4149       - \int_use:N \l_@@_initial_i_int
4150       - \int_eval:n { #2 - 1 }
4151     }
4152     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4153     {
4154       \int_set:Nn \l_@@_initial_j_int { #2 }
4155       \bool_set_true:N \l_@@_initial_open_bool
4156     }
4157   }
4158   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4159   {
4160     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4161     \bool_set_true:N \l_@@_final_open_bool
4162   }
4163   {
4164     \cs_if_exist:cTF
4165     {
4166       pgf @ sh @ ns @ \@@_env:
4167       - \int_use:N \l_@@_final_i_int
4168       - \int_eval:n { #2 + #3 }
4169     }
4170     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4171     {
4172       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4173       \bool_set_true:N \l_@@_final_open_bool
4174     }
4175   }
4176   \group_begin:
4177   \int_compare:nNnTF { #1 } = 0
4178   { \color { nicematrix-first-row } }
4179   {
4180     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4181     { \color { nicematrix-last-row } }
4182   }
4183   \keys_set:nn { NiceMatrix / xdots } { #4 }
4184   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4185   \@@_actually_draw_Ldots:
4186   \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).

```

4187     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4188     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4189   }

4190   \AtBeginDocument
4191   {
4192     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4193     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4194     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4195     {
4196       \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4197       {
4198         \@@_Vdotsfor:nnnn

```

```

4199         { \int_use:N \c@iRow }
4200         { \int_use:N \c@jCol }
4201         { #2 }
4202         {
4203             #1 , #3 ,
4204             down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4205         }
4206     }
4207 }
4208 }

```

Enf of \AtBeginDocument.

```

4209 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4210 {
4211     \bool_set_false:N \l_@@_initial_open_bool
4212     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

4213     \int_set:Nn \l_@@_initial_j_int { #2 }
4214     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

4215     \int_compare:nNnTF #1 = 1
4216     {
4217         \int_set:Nn \l_@@_initial_i_int 1
4218         \bool_set_true:N \l_@@_initial_open_bool
4219     }
4220     {
4221         \cs_if_exist:cTF
4222         {
4223             pgf @ sh @ ns @ \@@_env:
4224             - \int_eval:n { #1 - 1 }
4225             - \int_use:N \l_@@_initial_j_int
4226         }
4227         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4228         {
4229             \int_set:Nn \l_@@_initial_i_int { #1 }
4230             \bool_set_true:N \l_@@_initial_open_bool
4231         }
4232     }
4233     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4234     {
4235         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4236         \bool_set_true:N \l_@@_final_open_bool
4237     }
4238     {
4239         \cs_if_exist:cTF
4240         {
4241             pgf @ sh @ ns @ \@@_env:
4242             - \int_eval:n { #1 + #3 }
4243             - \int_use:N \l_@@_final_j_int
4244         }
4245         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4246         {
4247             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4248             \bool_set_true:N \l_@@_final_open_bool
4249         }
4250     }
4251     \group_begin:
4252     \int_compare:nNnTF { #2 } = 0
4253     { \color { nicematrix-first-col } }
4254     {
4255         \int_compare:nNnT { #2 } = \g_@@_col_total_int
4256         { \color { nicematrix-last-col } }

```

```

4257     }
4258     \keys_set:nn { NiceMatrix / xdots } { #4 }
4259     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4260     \@@_actually_draw_Vdots:
4261     \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).

```

4262     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4263     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4264 }

```

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

```

4265 \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).⁷¹

```

4266 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4267 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

4268 \AtBeginDocument
4269 {
4270     \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4271     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4272     \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4273     {
4274         \group_begin:
4275         \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4276         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4277         \use:e
4278         {
4279             \@@_line_i:nn
4280             { \@@_double_int_eval:n #2 \q_stop }
4281             { \@@_double_int_eval:n #3 \q_stop }
4282         }
4283         \group_end:
4284     }
4285 }
4286 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4287 {
4288     \bool_set_false:N \l_@@_initial_open_bool
4289     \bool_set_false:N \l_@@_final_open_bool
4290     \bool_if:nTF
4291     {

```

⁷¹Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

4292     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4293     ||
4294     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4295   }
4296   {
4297     \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4298   }
4299   { \@@_draw_line_ii:nn { #1 } { #2 } }
4300 }
4301 \AtBeginDocument
4302 {
4303   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4304   {

```

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:.`

```

4305     \c_@@_pgfortikzpicture_tl
4306     \@@_draw_line_iii:nn { #1 } { #2 }
4307     \c_@@_endpgfortikzpicture_tl
4308   }
4309 }

```

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

```

4310 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4311 {
4312   \pgfrememberpicturepositiononpagetrue
4313   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4314   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4315   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4316   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4317   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4318   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4319   \@@_draw_line:
4320 }

```

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).

The command `\RowStyle`

```

4321 \keys_define:nn { NiceMatrix / RowStyle }
4322 {
4323   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4324   cell-space-top-limit .initial:n = \c_zero_dim ,
4325   cell-space-top-limit .value_required:n = true ,
4326   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4327   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4328   cell-space-bottom-limit .value_required:n = true ,
4329   cell-space-limits .meta:n =
4330   {
4331     cell-space-top-limit = #1 ,
4332     cell-space-bottom-limit = #1 ,
4333   } ,
4334   color .tl_set:N = \l_tmpa_tl ,
4335   color .value_required:n = true ,
4336   bold .bool_set:N = \l_tmpa_bool ,
4337   bold .default:n = true ,
4338   bold .initial:n = false ,
4339   nb-rows .int_set:N = \l_@@_key_nb_rows_int ,
4340   nb-rows .value_required:n = true ,

```

```

4341     nb-rows .initial:n = 1 ,
4342     rowcolor .tl_set:N = \l_tmpc_tl ,
4343     rowcolor .value_required:n = true ,
4344     rowcolor .initial:n = \c_empty_tl ,
4345     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
4346 }

```

```

4347 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4348 {
4349     \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

4350     \tl_if_empty:NF \l_tmpc_tl
4351     {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

4352         \tl_gput_right:Nx \g_nicematrix_code_before_tl
4353         {
4354             \@@_rectanglecolor
4355             { \l_tmpc_tl }
4356             { \int_use:N \c@iRow - \int_use:N \c@jCol }
4357             { \int_use:N \c@iRow - * }
4358         }

```

Then, the other rows (if there is several rows).

```

4359         \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4360         {
4361             \tl_gput_right:Nx \g_nicematrix_code_before_tl
4362             {
4363                 \@@_rowcolor
4364                 { \l_tmpc_tl }
4365                 {
4366                     \int_eval:n { \c@iRow + 1 }
4367                     - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4368                 }
4369             }
4370         }
4371     }
4372     \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4373     \tl_gput_right:Nx \g_@@_row_style_tl
4374     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4375     \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

4376     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4377     {
4378         \tl_gput_right:Nx \g_@@_row_style_tl
4379         {
4380             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4381             {
4382                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
4383                 { \dim_use:N \l_tmpa_dim }
4384             }
4385         }
4386     }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

4387     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4388     {
4389         \tl_gput_right:Nx \g_@@_row_style_tl
4390         {
4391             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4392             {
4393                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4394                 { \dim_use:N \l_tmpb_dim }
4395             }

```

```

4396     }
4397   }
\l_tmpa_tl is the value of the key color of \RowStyle.
4398   \tl_if_empty:NF \l_tmpa_tl
4399   {
4400     \tl_gput_right:Nx \g_@@_row_style_tl
4401     { \mode_leave_vertical: \exp_not:N \color { \l_tmpa_tl } }
4402   }
\l_tmpa_bool is the value of the key bold.
4403   \bool_if:NT \l_tmpa_bool
4404   {
4405     \tl_gput_right:Nn \g_@@_row_style_tl
4406     {
4407       \if_mode_math:
4408         \c_math_toggle_token
4409         \bfseries \boldmath
4410         \c_math_toggle_token
4411       \else:
4412         \bfseries \boldmath
4413       \fi:
4414     }
4415   }
4416   \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4417   \g_@@_row_style_tl
4418   \ignorespaces
4419 }

```

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`, `\@@_rectanglecolor` and `\@@_rowlistcolors` don’t directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence `\g_@@_colors_seq` will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: `[gray]{0.5}`).
- For the color whose index in `\g_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\g_@@_color_i_tl`. In that token list, the instructions will be written using `\@@_cartesian_color:nn` and `\@@_rectanglecolor:nn`.

`#1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_colors_seq:nn` doesn’t only add a color to `\g_@@_colors_seq`: it also updates the corresponding token list `\g_@@_color_i_tl`. We add in a global way because the final user may use the instructions such as `\cellcolor` in a loop of `pgffor` in the `\CodeBefore` (and we recall that a loop of `pgffor` is encapsulated in a group).

```

4420 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4421 {

```

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.

```

4422   \int_zero:N \l_tmpa_int
4423   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4424   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4425   \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

4426 {
4427   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4428   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4429 }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

4430 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4431 }

4432 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4433 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x x }

```

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`).

```

4434 \cs_new_protected:Npn \@@_actually_color:
4435 {
4436   \pgfpicture
4437   \pgf@relevantforpicturesizefalse
4438   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4439   {
4440     \color ##2
4441     \use:c { g_@@_color _ ##1 _tl }
4442     \tl_gclear:c { g_@@_color _ ##1 _tl }
4443     \pgfusepath { fill }
4444   }
4445   \endpgfpicture
4446 }

4447 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4448 {
4449   \tl_set:Nn \l_@@_rows_tl { #1 }
4450   \tl_set:Nn \l_@@_cols_tl { #2 }
4451   \@@_cartesian_path:
4452 }

```

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

```

4453 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4454 {
4455   \tl_if_blank:nF { #2 }
4456   {
4457     \@@_add_to_colors_seq:xn
4458     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4459     { \@@_cartesian_color:nn { #3 } { - } }
4460   }
4461 }

```

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

```

4462 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4463 {
4464   \tl_if_blank:nF { #2 }
4465   {
4466     \@@_add_to_colors_seq:xn
4467     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4468     { \@@_cartesian_color:nn { - } { #3 } }
4469   }
4470 }

```


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

```

4471 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4472 {
4473   \tl_if_blank:nF { #2 }
4474   {
4475     \@@_add_to_colors_seq:xn
4476     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4477     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4478   }
4479 }

```

The last argument is the radius of the corners of the rectangle.

```

4480 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4481 {
4482   \tl_if_blank:nF { #2 }
4483   {
4484     \@@_add_to_colors_seq:xn
4485     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4486     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4487   }
4488 }

```

The last argument is the radius of the corners of the rectangle.

```

4489 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4490 {
4491   \@@_cut_on_hyphen:w #1 \q_stop
4492   \tl_clear_new:N \l_tmpc_tl
4493   \tl_clear_new:N \l_tmpd_tl
4494   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
4495   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
4496   \@@_cut_on_hyphen:w #2 \q_stop
4497   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
4498   \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`.

```

4499   \@@_cartesian_path:n { #3 }
4500 }

```

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

```

4501 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4502 {
4503   \clist_map_inline:nn { #3 }
4504   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4505 }

```

```

4506 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4507 {
4508   \int_step_inline:nn { \int_use:N \c@iRow }
4509   {
4510     \int_step_inline:nn { \int_use:N \c@jCol }
4511     {
4512       \int_if_even:nTF { ####1 + ##1 }
4513       { \@@_cellcolor [ #1 ] { #2 } }
4514       { \@@_cellcolor [ #1 ] { #3 } }
4515       { ##1 - ####1 }
4516     }
4517   }
4518 }

```

```

4519 \keys_define:nn { NiceMatrix / arraycolor }
4520 { except-corners .code:n = \@@_error:n { key~except-corners } }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns). The third argument is a optional argument which a list of pairs key-value.

```

4521 \NewDocumentCommand \@@_arraycolor { 0 { } m 0 { } }
4522 {
4523   \keys_set:nn { NiceMatrix / arraycolor } { #3 }
4524   \@@_rectanglecolor [ #1 ] { #2 }
4525   { 1 - 1 }
4526   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4527 }

```

```

4528 \keys_define:nn { NiceMatrix / rowcolors }
4529 {
4530   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4531   respect-blocks .default:n = true ,
4532   cols .tl_set:N = \l_@@_cols_tl ,
4533   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4534   restart .default:n = true ,
4535   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4536 }

```

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 list of colors ; **#4** is for the optional list of pairs key-value.

```

4537 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
4538 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

4539   \group_begin:
4540   \seq_clear_new:N \l_@@_colors_seq
4541   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
4542   \tl_clear_new:N \l_@@_cols_tl
4543   \tl_set:Nn \l_@@_cols_tl { - }
4544   \keys_set:nn { NiceMatrix / rowcolors } { #4 }

```

The counter `\l_@@_color_int` will be the rank of the current color in the list of colors (modulo the length of the list).

```

4545   \int_zero_new:N \l_@@_color_int
4546   \int_set:Nn \l_@@_color_int 1
4547   \bool_if:NT \l_@@_respect_blocks_bool
4548   {

```

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`).

```

4549     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4550     \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
4551     { \@@_not_in_exterior_p:nnnnn ##1 }
4552   }
4553   \pgfpicture
4554   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

4555   \clist_map_inline:nn { #2 }
4556   {
4557     \tl_set:Nn \l_tmpa_tl { ##1 }

```

```

4558 \tl_if_in:NnTF \l_tmpa_tl { - }
4559 { \@@_cut_on_hyphen:w ##1 \q_stop }
4560 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c{iRow} } }

```

Now, `\l_tmpa_tl` and `\l_tmpb_tl` are the first row and the last row of the interval of rows that we have to treat. The counter `\l_tmpa_int` will be the index of the loop over the rows.

```

4561 \int_set:Nn \l_tmpa_int \l_tmpa_tl
4562 \bool_if:NTF \l_@@_rowcolors_restart_bool
4563 { \int_set:Nn \l_@@_color_int 1 }
4564 { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
4565 \int_zero_new:N \l_tmpc_int
4566 \int_set:Nn \l_tmpc_int \l_tmpb_tl
4567 \int_do_until:nNnn \l_tmpa_int > \l_tmpc_int
4568 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4569 \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).

```

4570 \bool_if:NT \l_@@_respect_blocks_bool
4571 {
4572   \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
4573   { \@@_intersect_our_row_p:nnnnn #####1 }
4574   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4575 }
4576 \tl_set:Nx \l_@@_rows_tl
4577 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_tmpc_tl` will be the color that we will use.

```

4578 \tl_clear_new:N \l_@@_color_tl
4579 \tl_set:Nx \l_@@_color_tl
4580 {
4581   \@@_color_index:n
4582   {
4583     \int_mod:nn
4584     { \l_@@_color_int - 1 }
4585     { \seq_count:N \l_@@_colors_seq }
4586     + 1
4587   }
4588 }
4589 \tl_if_empty:NF \l_@@_color_tl
4590 {
4591   \@@_add_to_colors_seq:xx
4592   { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
4593   { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
4594 }
4595 \int_incr:N \l_@@_color_int
4596 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4597 }
4598 }
4599 \endpgfpicture
4600 \group_end:
4601 }

```

The command `\@@_color_index:n` peeks in `\l_@@_colors_seq` the color at the index `#1`. However, if that color is the symbol `=`, the previous one is poken. This macro is recursive.

```

4602 \cs_new:Npn \@@_color_index:n #1
4603 {
4604   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
4605   { \@@_color_index:n { #1 - 1 } }
4606   { \seq_item:Nn \l_@@_colors_seq { #1 } }
4607 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

4608 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4609 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

4610 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
4611 {
4612   \int_compare:nNnT { #3 } > \l_tmpb_int
4613   { \int_set:Nn \l_tmpb_int { #3 } }
4614 }

4615 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
4616 {
4617   \bool_lazy_or:nnTF
4618   { \int_compare_p:nNn { #4 } = \c_zero_int }
4619   { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
4620   \prg_return_false:
4621   \prg_return_true:
4622 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

4623 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
4624 {
4625   \bool_if:nTF
4626   {
4627     \int_compare_p:n { #1 <= \l_tmpa_int }
4628     &&
4629     \int_compare_p:n { \l_tmpa_int <= #3 }
4630   }
4631   \prg_return_true:
4632   \prg_return_false:
4633 }

```

The following command uses two implicit arguments: `\l_@@_rows_tl` and `\l_@@_cols_tl` which are specifications for a set of rows and a set of columns. It creates a path but does *not* fill it. It must be filled by another command after. The argument is the radius of the corners. We define below a command `\@@_cartesian_path:` which corresponds to a value 0 pt for the radius of the corners. This command is in particular used in `\@@_rectanglecolor:nnn` (used in `\@@_rectanglecolor`, itself used in `\@@_cellcolor`).

```

4634 \cs_new_protected:Npn \@@_cartesian_path:n #1
4635 {
4636   \bool_lazy_and:nnT
4637   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4638   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4639   {
4640     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4641     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4642   }

```

We begin the loop over the columns.

```

4643 \clist_map_inline:Nn \l_@@_cols_tl
4644 {
4645   \tl_set:Nn \l_tmpa_tl { ##1 }
4646   \tl_if_in:NnTF \l_tmpa_tl { - }
4647   { \@@_cut_on_hyphen:w ##1 \q_stop }
4648   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4649   \bool_lazy_or:nnT
4650   { \tl_if_blank_p:V \l_tmpa_tl }
4651   { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4652   { \tl_set:Nn \l_tmpa_tl { 1 } }

```

```

4653 \bool_lazy_or:nnT
4654 { \tl_if_blank_p:V \l_tmpb_tl }
4655 { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4656 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4657 \int_compare:nNnT \l_tmpb_tl > \c@jCol
4658 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_tmpc_tl` will contain the number of column.

```

4659 \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl

```

If we decide to provide the commands `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors` and `\chessboardcolors` in the code-before of a `\SubMatrix`, we will have to modify the following line, by adding a kind of offset. We will have also some other lines to modify.

```

4660 \@@_qpoint:n { col - \l_tmpa_tl }
4661 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4662 { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4663 { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4664 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4665 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4666 \clist_map_inline:Nn \l_@@_rows_tl
4667 {
4668   \tl_set:Nn \l_tmpa_tl { #####1 }
4669   \tl_if_in:NnTF \l_tmpa_tl { - }
4670   { \@@_cut_on_hyphen:w #####1 \q_stop }
4671   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4672   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4673   \tl_if_empty:NT \l_tmpb_tl
4674   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4675   \int_compare:nNnT \l_tmpb_tl > \c@iRow
4676   { \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`.

```

4677 \seq_if_in:NxF \l_@@_corners_cells_seq
4678 { \l_tmpa_tl - \l_tmpc_tl }
4679 {
4680   \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
4681   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4682   \@@_qpoint:n { row - \l_tmpa_tl }
4683   \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4684   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4685   \pgfpathrectanglecorners
4686   { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4687   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4688 }
4689 }
4690 }
4691 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands `\@@_rowcolors`, `\@@_columncolor` and `\@@_rowcolor:n` (used in `\@@_rowcolor`).

```

4692 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to 2,4-6,8-* and `\c@jCol` equal to 10, the `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

4693 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4694 {
4695   \clist_set_eq:NN \l_tmpa_clist #1
4696   \clist_clear:N #1
4697   \clist_map_inline:Nn \l_tmpa_clist
4698   {

```

```

4699 \tl_set:Nn \l_tmpa_tl { ##1 }
4700 \tl_if_in:NnTF \l_tmpa_tl { - }
4701 { \@@_cut_on_hyphen:w ##1 \q_stop }
4702 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4703 \bool_lazy_or:nnT
4704 { \tl_if_blank_p:V \l_tmpa_tl }
4705 { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4706 { \tl_set:Nn \l_tmpa_tl { 1 } }
4707 \bool_lazy_or:nnT
4708 { \tl_if_blank_p:V \l_tmpb_tl }
4709 { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4710 { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4711 \int_compare:nNnT \l_tmpb_tl > #2
4712 { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4713 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4714 { \clist_put_right:Nn #1 { ####1 } }
4715 }
4716 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```

4717 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4718 {
4719   \peek_remove_spaces:n
4720   {
4721     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4722     {

```

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`).

```

4723       \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
4724       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4725     }
4726   }
4727 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the tabular.

```

4728 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4729 {
4730   \peek_remove_spaces:n
4731   {
4732     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4733     {
4734       \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4735       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4736       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4737     }
4738   }
4739 }

```

```

4740 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4741 {

```

With the following line, we test whether the cell is the first one we encounter in its column (don't forget that some rows may be incomplete).

```

4742   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4743   {

```

You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells). Be careful: maybe this is not effective since we have an analyze of the instructions in the `\CodeBefore` in order to fill color by color (to avoid the thin white lines).

```

4744 \tl_gput_left:Nx \g_nicematrix_code_before_tl
4745 {
4746   \exp_not:N \columncolor [ #1 ]
4747   { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4748 }
4749 }
4750 }

```

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`.

```

4751 \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`.

```

4752 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4753 {
4754   \int_compare:nNnTF \l_@@_first_col_int = 0
4755   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4756   {
4757     \int_compare:nNnTF \c@jCol = 0
4758     {
4759       \int_compare:nNnF \c@iRow = { -1 }
4760       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4761     }
4762     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4763   }
4764 }

```

This definition may seem complicated but 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.

```

4765 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4766 {
4767   \int_compare:nNnF \c@iRow = 0
4768   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4769 }

```

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 `|`. `#3` and `#4` are the numbers of rows that define the delimitation of the horizontal rule that we have to draw. If `#4` is empty, that means that the rule extends until the last row.

```

4770 \cs_new_protected:Npn \@@_vline:nnnn #1 #2 #3 #4
4771 {

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

4772 \int_compare:nNnT { #1 } < { \c@jCol + 2 }
4773 {
4774   \pgfpicture
4775   \@@_vline_i:nnnn { #1 } { #2 } { #3 } { #4 }
4776   \endpgfpicture
4777 }
4778 }
4779 \cs_new_protected:Npn \@@_vline_i:nnnn #1 #2 #3 #4
4780 {

```

`\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_tmppc_tl`.

```

4781 \tl_set:Nx \l_tmpb_tl { #1 }
4782 \tl_clear_new:N \l_tmppc_tl
4783 \int_step_variable:nnNn
4784 { #3 }
4785 { \tl_if_blank:nTF { #4 } { \int_use:N \c@iRow } { #4 } }
4786 \l_tmpa_tl
4787 {

```

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.

```

4788 \bool_gset_true:N \g_tmpa_bool
4789 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4790 { \@@_test_vline_in_block:nnnnn ##1 }
4791 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4792 { \@@_test_vline_in_block:nnnnn ##1 }
4793 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4794 { \@@_test_vline_in_stroken_block:nnnn ##1 }
4795 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
4796 \bool_if:NTF \g_tmpa_bool
4797 {
4798   \tl_if_empty:NT \l_tmppc_tl

```

We keep in memory that we have a rule to draw.

```

4799 { \tl_set_eq:NN \l_tmppc_tl \l_tmpa_tl }
4800 }
4801 {
4802   \tl_if_empty:NF \l_tmppc_tl
4803   {
4804     \@@_vline_ii:nnnn
4805     { #1 }
4806     { #2 }
4807     \l_tmppc_tl
4808     { \int_eval:n { \l_tmpa_tl - 1 } }
4809     \tl_clear:N \l_tmppc_tl
4810   }
4811 }
4812 }
4813 \tl_if_empty:NF \l_tmppc_tl
4814 {
4815   \@@_vline_ii:nnnn
4816   { #1 }
4817   { #2 }
4818   \l_tmppc_tl
4819   { \tl_if_blank:nTF { #4 } { \int_use:N \c@iRow } { #4 } }
4820   \tl_clear:N \l_tmppc_tl
4821 }
4822 }

```



```

4823 \cs_new_protected:Npn \@@_test_in_corner_v:
4824 {
4825   \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
4826   {
4827     \seq_if_in:NxT
4828     \l_@@_corners_cells_seq
4829     { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4830     { \bool_set_false:N \g_tmpa_bool }
4831   }
4832   {
4833     \seq_if_in:NxT
4834     \l_@@_corners_cells_seq
4835     { \l_tmpa_tl - \l_tmpb_tl }
4836     {
4837       \int_compare:nNnTF \l_tmpb_tl = 1
4838       { \bool_set_false:N \g_tmpa_bool }
4839       {
4840         \seq_if_in:NxT
4841         \l_@@_corners_cells_seq
4842         { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4843         { \bool_set_false:N \g_tmpa_bool }
4844       }
4845     }
4846   }
4847 }

```

#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.

```

4848 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
4849 {
4850   \bool_if:NTF \l_@@_dotted_bool
4851   { \@@_vline_iv:nnn { #1 } { #3 } { #4 } }
4852   { \@@_vline_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
4853 }

```

The following code is for the standard case (the rule which is drawn is a solid rule).

#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.

```

4854 \cs_new_protected:Npn \@@_vline_iii:nnnn #1 #2 #3 #4
4855 {
4856   \pgfrememberpicturepositiononpagetrue
4857   \pgf@relevantforpicturesizefalse
4858   \@@_qpoint:n { row - #3 }
4859   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4860   \@@_qpoint:n { col - #1 }
4861   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4862   \@@_qpoint:n { row - \@@_succ:n { #4 } }
4863   \dim_set_eq:NN \l_tmpc_dim \pgf@y
4864   \bool_lazy_all:nT
4865   {
4866     { \int_compare_p:nNn { #2 } > 1 }
4867     { \cs_if_exist_p:N \CT@drsc@ }
4868     { ! \tl_if_blank_p:V \CT@drsc@ }
4869   }
4870   {
4871     \group_begin:
4872     \CT@drsc@
4873     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4874     \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
4875     \dim_set:Nn \l_tmpd_dim
4876     { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4877     \pgfpathrectanglecorners

```

```

4878     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4879     { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4880     \pgfusepath { fill }
4881     \group_end:
4882 }
4883 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4884 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4885 \prg_replicate:nn { #2 - 1 }
4886 {
4887     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4888     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4889     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4890     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4891 }
4892 \CT@arc@
4893 \pgfsetlinewidth { 1.1 \arrayrulewidth }
4894 \pgfsetrectcap
4895 \pgfusepathqstroke
4896 }

```

The following code is for the case of a dotted rule (with our system).

#1 is the number of the column; **#2** and **#3** are the numbers of the rows between which the rule has to be drawn.

```

4897 \cs_new_protected:Npn \@@_vline_iv:nnn #1 #2 #3
4898 {
4899     \pgfrememberpicturepositiononpagetrue
4900     \pgf@relevantforpicturesizefalse
4901     \@@_qpoint:n { col - #1 }
4902     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4903     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4904     \@@_qpoint:n { row - #2 }
4905     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4906     \@@_qpoint:n { row - \@@_succ:n { #3 } }
4907     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4908     \@@_draw_line:
4909 }

```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

4910 \cs_new_protected:Npn \@@_draw_vlines:
4911 {
4912     \int_step_inline:nnn
4913     {
4914         \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4915         1 2
4916     }
4917     {
4918         \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4919         { \@@_succ:n \c@jCol }
4920         \c@jCol
4921     }
4922     {
4923         \tl_if_eq:NnF \l_@@_vlines_clist { all }
4924         { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4925         { \@@_vline:nnnn { ##1 } 1 1 { } }
4926     }
4927 }

```

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`. #3 and #4 are numbers of columns that define the delimitation of the horizontal rule that we have to draw. If #4 is empty, that means that the rule extends until the last column.

```

4928 \cs_new_protected:Npn \@@_hline:nnnn #1 #2 #3 #4
4929 {
4930   \pgfpicture
4931   \@@_hline_i:nnnn { #1 } { #2 } { #3 } { #4 }
4932   \endpgfpicture
4933 }
4934 \cs_new_protected:Npn \@@_hline_i:nnnn #1 #2 #3 #4
4935 {

```

`\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`.

```

4936   \tl_set:Nn \l_tmpa_tl { #1 }
4937   \tl_clear_new:N \l_tmpc_tl
4938   \int_step_variable:nnNn
4939     { #3 }
4940     { \tl_if_blank:nTF { #4 } { \int_use:N \c@jCol } { #4 } }
4941     \l_tmpb_tl
4942   {

```

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.

```

4943     \bool_gset_true:N \g_tmpa_bool
4944     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4945       { \@@_test_hline_in_block:nnnn ##1 }
4946     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4947       { \@@_test_hline_in_block:nnnn ##1 }
4948     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4949       { \@@_test_hline_in_stroken_block:nnnn ##1 }
4950     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4951     \bool_if:NTF \g_tmpa_bool
4952     {
4953       \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4954       { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
4955     }
4956     {
4957       \tl_if_empty:NF \l_tmpc_tl
4958       {
4959         \@@_hline_ii:nnnn
4960         { #1 }
4961         { #2 }
4962         \l_tmpc_tl
4963         { \int_eval:n { \l_tmpb_tl - 1 } }
4964         \tl_clear:N \l_tmpc_tl
4965       }
4966     }
4967   }
4968   \tl_if_empty:NF \l_tmpc_tl
4969   {
4970     \@@_hline_ii:nnnn
4971     { #1 }
4972     { #2 }
4973     \l_tmpc_tl

```

```

4974         { \tl_if_blank:nTF { #4 } { \int_use:N \c@jCol } { #4 } }
4975     \tl_clear:N \l_tmpc_tl
4976 }
4977 }

```

```

4978 \cs_new_protected:Npn \@@_test_in_corner_h:
4979 {
4980     \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
4981     {
4982         \seq_if_in:NxT
4983         \l_@@_corners_cells_seq
4984         { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4985         { \bool_set_false:N \g_tmpa_bool }
4986     }
4987     {
4988         \seq_if_in:NxT
4989         \l_@@_corners_cells_seq
4990         { \l_tmpa_tl - \l_tmpb_tl }
4991         {
4992             \int_compare:nNnTF \l_tmpa_tl = 1
4993             { \bool_set_false:N \g_tmpa_bool }
4994             {
4995                 \seq_if_in:NxT
4996                 \l_@@_corners_cells_seq
4997                 { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4998                 { \bool_set_false:N \g_tmpa_bool }
4999             }
5000         }
5001     }
5002 }

```

```

5003 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
5004 {
5005     \bool_if:NTF \l_@@_dotted_bool
5006     { \@@_hline_iv:nnn { #1 } { #3 } { #4 } }
5007     { \@@_hline_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
5008 }

```

#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.

```

5009 \cs_new_protected:Npn \@@_hline_iii:nnnn #1 #2 #3 #4
5010 {
5011     \pgfrememberpicturerepositiononpagetrue
5012     \pgf@relevantforpicturesizefalse
5013     \@@_qpoint:n { col - #3 }
5014     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5015     \@@_qpoint:n { row - #1 }
5016     \dim_set_eq:NN \l_tmpb_dim \pgf@y
5017     \@@_qpoint:n { col - \@@_succ:n { #4 } }
5018     \dim_set_eq:NN \l_tmpc_dim \pgf@x
5019     \bool_lazy_all:nT
5020     {
5021         { \int_compare_p:nNn { #2 } > 1 }
5022         { \cs_if_exist_p:N \CT@drsc@ }
5023         { ! \tl_if_blank_p:V \CT@drsc@ }
5024     }
5025     {
5026         \group_begin:
5027         \CT@drsc@
5028         \dim_set:Nn \l_tmpd_dim
5029         { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }

```

```

5030     \pgfpathrectanglecorners
5031     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5032     { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
5033     \pgfusepathqfill
5034     \group_end:
5035 }
5036 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5037 \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5038 \prg_replicate:nn { #2 - 1 }
5039 {
5040     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5041     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5042     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5043     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5044 }
5045 \CT@arc@
5046 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5047 \pgfsetrectcap
5048 \pgfusepathqstroke
5049 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). 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}

```

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

```

5050 \cs_new_protected:Npn \@@_hline_iv:nnn #1 #2 #3
5051 {
5052     \pgfrememberpicturepositiononpagetrue
5053     \pgf@relevantforpicturesizefalse
5054     \@@_qpoint:n { row - #1 }
5055     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5056     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5057     \@@_qpoint:n { col - #2 }
5058     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5059     \int_compare:nNnT { #2 } = 1
5060     {
5061         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5062         \bool_if:NT \l_@@_NiceArray_bool
5063         { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }

```

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.

```

5064     \tl_if_eq:NnF \g_@@_left_delim_tl (
5065     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
5066     )
5067     \@@_qpoint:n { col - \@@_succ:n { #3 } }
5068     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5069     \int_compare:nNnT { #3 } = \c@jCol
5070     {

```

```

5071 \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5072 \bool_if:NT \l_@@_NiceArray_bool
5073 { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5074 \tl_if_eq:NnF \g_@@_right_delim_tl )
5075 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
5076 }
5077 \@@_draw_line:
5078 }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners (if the key `corners` is used).

```

5079 \cs_new_protected:Npn \@@_draw_hlines:
5080 {
5081   \int_step_inline:nnn
5082   {
5083     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5084     1 2
5085   }
5086   {
5087     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5088     { \@@_succ:n \c@iRow }
5089     \c@iRow
5090   }
5091   {
5092     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5093     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5094     { \@@_hline:nnnn { ##1 } 1 1 { } }
5095   }
5096 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5097 \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.

```

5098 \cs_set:Npn \@@_Hline_i:n #1
5099 {
5100   \peek_meaning_ignore_spaces:NTF \Hline
5101   { \@@_Hline_ii:nn { #1 + 1 } }
5102   { \@@_Hline_iii:n { #1 } }
5103 }
5104 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5105 \cs_set:Npn \@@_Hline_iii:n #1
5106 {
5107   \skip_vertical:n
5108   {
5109     \arrayrulewidth * ( #1 )
5110     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
5111   }
5112   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5113   { \@@_hline:nnnn { \@@_succ:n { \c@iRow } } { #1 } 1 { } }
5114   \ifnum 0 = ` { \fi }
5115 }

```

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`.

```

5116 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4
5117 {
5118   \bool_lazy_all:nT
5119   {
5120     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
5121     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5122     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5123     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5124   }
5125   { \bool_gset_false:N \g_tmpa_bool }
5126 }

```

The same for vertical rules.

```

5127 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4
5128 {
5129   \bool_lazy_all:nT
5130   {
5131     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5132     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5133     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
5134     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5135   }
5136   { \bool_gset_false:N \g_tmpa_bool }
5137 }
5138 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
5139 {
5140   \bool_lazy_all:nT
5141   {
5142     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5143     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
5144     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5145     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5146   }
5147   { \bool_gset_false:N \g_tmpa_bool }
5148 }
5149 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
5150 {
5151   \bool_lazy_all:nT
5152   {
5153     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5154     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5155     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5156     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
5157   }
5158   { \bool_gset_false:N \g_tmpa_bool }
5159 }

```

The key corners

When the key `corners` is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

```

5160 \cs_new_protected:Npn \@@_compute_corners:
5161 {

```

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

```

5162   \seq_clear_new:N \l_@@_corners_cells_seq
5163   \clist_map_inline:Nn \l_@@_corners_clist
5164   {
5165     \str_case:nnF { ##1 }

```

```

5166     {
5167         { NW }
5168         { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
5169         { NE }
5170         { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
5171         { SW }
5172         { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
5173         { SE }
5174         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
5175     }
5176     { \@@_error:nn { bad~corner } { ##1 } }
5177 }

```

Even if the user has used the key `corners` (or the key `hvlines-except-corners`), the list of cells in the corners may be empty.

```

5178 \seq_if_empty:NF \l_@@_corners_cells_seq
5179 {

```

You write on the `aux` file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which color the `rows`, `columns` and `cells` must not color the cells in the corners.

```

5180 \tl_gput_right:Nx \g_@@_aux_tl
5181 {
5182     \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
5183     { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
5184 }
5185 }
5186 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_corners_cells_seq`.

The six arguments of `\@@_compute_a_corner:nnnnnn` are as follow:

- **#1** and **#2** are the number of row and column of the cell which is actually in the corner;
- **#3** and **#4** are the steps in rows and the step in columns when moving from the corner;
- **#5** is the number of the final row when scanning the rows from the corner;
- **#6** is the number of the final column when scanning the columns from the corner.

```

5187 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
5188 {

```

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.

```

5189 \bool_set_false:N \l_tmpa_bool
5190 \int_zero_new:N \l_@@_last_empty_row_int
5191 \int_set:Nn \l_@@_last_empty_row_int { #1 }
5192 \int_step_inline:nnnn { #1 } { #3 } { #5 }
5193 {
5194     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
5195     \bool_lazy_or:nnTF
5196     {
5197         \cs_if_exist_p:c
5198         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
5199     }
5200     \l_tmpb_bool
5201     { \bool_set_true:N \l_tmpa_bool }
5202     {
5203         \bool_if:NF \l_tmpa_bool

```



```

5204         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
5205     }
5206 }

```

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

```

5207 \bool_set_false:N \l_tmpa_bool
5208 \int_zero_new:N \l_@@_last_empty_column_int
5209 \int_set:Nn \l_@@_last_empty_column_int { #2 }
5210 \int_step_inline:nnnn { #2 } { #4 } { #6 }
5211 {
5212     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
5213     \bool_lazy_or:nnTF
5214         \l_tmpb_bool
5215         {
5216             \cs_if_exist_p:c
5217                 { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
5218         }
5219         { \bool_set_true:N \l_tmpa_bool }
5220         {
5221             \bool_if:NF \l_tmpa_bool
5222                 { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
5223         }
5224     }

```

Now, we loop over the rows.

```

5225 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
5226 {

```

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

```

5227 \bool_set_false:N \l_tmpa_bool
5228 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
5229 {
5230     \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
5231     \bool_lazy_or:nnTF
5232         \l_tmpb_bool
5233         {
5234             \cs_if_exist_p:c
5235                 { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
5236         }
5237         { \bool_set_true:N \l_tmpa_bool }
5238         {
5239             \bool_if:NF \l_tmpa_bool
5240             {
5241                 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
5242                 \seq_put_right:Nn
5243                     \l_@@_corners_cells_seq
5244                     { ##1 - #####1 }
5245             }
5246         }
5247     }
5248 }
5249 }

```

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`).

```

5250 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
5251 {
5252     \int_set:Nn \l_tmpa_int { #1 }
5253     \int_set:Nn \l_tmpb_int { #2 }
5254     \bool_set_false:N \l_tmpb_bool
5255     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5256         { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
5257 }

```

```

5258 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
5259 {
5260   \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
5261   {
5262     \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
5263     {
5264       \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
5265       {
5266         \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
5267         { \bool_set_true:N \l_tmpb_bool }
5268       }
5269     }
5270   }
5271 }

```

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`.

```

5272 \cs_new:Npn \@@_hdottedline:
5273 {
5274   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
5275   \@@_hdottedline_i:
5276 }

```

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

```

5277 \cs_new_protected:Npn \@@_hdottedline_i:
5278 {

```

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).

```

5279   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5280   { \@@_hdottedline:n { \int_use:N \c@iRow } }
5281 }

```

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.

```

5282 \cs_new_protected:Npn \@@_hdottedline:n #1
5283 {
5284   \group_begin:
5285   \bool_set_true:N \l_@@_dotted_bool
5286   \@@_hline:nnnn { #1 } { 1 } { 1 } { \int_use:N \c@jCol }
5287   \group_end:
5288 }

```

Vertical dotted lines

```

5289 \cs_new_protected:Npn \@@_vdottedline:n #1
5290 {
5291   \group_begin:
5292   \bool_set_true:N \l_@@_dotted_bool
5293   \@@_vline:nnnn { \int_eval:n { #1 + 1 } } { 1 } { 1 } { \int_use:N \c@iRow }
5294   \group_end:
5295 }

```

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.

```
5296 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
5297 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5298 {
5299   auto-columns-width .code:n =
5300   {
5301     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5302     \dim_gzero_new:N \g_@@_max_cell_width_dim
5303     \bool_set_true:N \l_@@_auto_columns_width_bool
5304   }
5305 }

5306 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
5307 {
5308   \int_gincr:N \g_@@_NiceMatrixBlock_int
5309   \dim_zero:N \l_@@_columns_width_dim
5310   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5311   \bool_if:NT \l_@@_block_auto_columns_width_bool
5312   {
5313     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5314     {
5315       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
5316       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5317     }
5318   }
5319 }
```

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).

```
5320 {
5321   \bool_if:NT \l_@@_block_auto_columns_width_bool
5322   {
5323     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5324     \iow_shipout:Nx \@mainaux
5325     {
5326       \cs_gset:cpn
5327       { @@_max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```
5328       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5329     }
5330     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5331   }
5332 }
```

The extra nodes

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

```
5333 \cs_generate_variant:Nn \dim_min:nn { v n }
5334 \cs_generate_variant:Nn \dim_max:nn { v n }
```

The following command is called in \@@_use_arraybox_with_notes_c: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).

```

5335 \cs_new_protected:Npn \@@_create_extra_nodes:
5336 {
5337   \bool_if:NTF \l_@@_medium_nodes_bool
5338   {
5339     \bool_if:NTF \l_@@_large_nodes_bool
5340     \@@_create_medium_and_large_nodes:
5341     \@@_create_medium_nodes:
5342   }
5343   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5344 }

```

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`.

```

5345 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5346 {
5347   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5348   {
5349     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
5350     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
5351     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
5352     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
5353   }
5354   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5355   {
5356     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
5357     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
5358     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
5359     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
5360   }

```

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

```

5361   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5362   {
5363     \int_step_variable:nnNn
5364     \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.

```

5365     {
5366       \cs_if_exist:cT
5367       { 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`.

```

5368     {
5369       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5370       \dim_set:cn { l_@@_row\_@@_i: _min_dim }

```

```

5371         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
5372     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5373     {
5374         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
5375         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
5376     }

```

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`.

```

5377     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
5378     \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
5379     { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
5380     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5381     {
5382         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
5383         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
5384     }
5385 }
5386 }
5387 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

5388     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5389     {
5390         \dim_compare:nNnT
5391         { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
5392         {
5393             \@@_qpoint:n { row - \@@_i: - base }
5394             \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
5395             \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
5396         }
5397     }
5398     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5399     {
5400         \dim_compare:nNnT
5401         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
5402         {
5403             \@@_qpoint:n { col - \@@_j: }
5404             \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
5405             \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
5406         }
5407     }
5408 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

5409 \cs_new_protected:Npn \@@_create_medium_nodes:
5410 {
5411     \pgfpicture
5412     \pgfrememberpicturepositiononpagetrue
5413     \pgf@relevantforpicturesizefalse
5414     \@@_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”.

```

5415     \tl_set:Nn \l_@@_suffix_tl { -medium }
5416     \@@_create_nodes:
5417     \endpgfpicture
5418 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁷². 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:`.

```

5419 \cs_new_protected:Npn \@@_create_large_nodes:
5420 {
5421   \pgfpicture
5422     \pgfrememberpicturepositiononpagetrue
5423     \pgf@relevantforpicturesizefalse
5424     \@@_computations_for_medium_nodes:
5425     \@@_computations_for_large_nodes:
5426     \tl_set:Nn \l_@@_suffix_tl { - large }
5427     \@@_create_nodes:
5428   \endpgfpicture
5429 }

5430 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5431 {
5432   \pgfpicture
5433     \pgfrememberpicturepositiononpagetrue
5434     \pgf@relevantforpicturesizefalse
5435     \@@_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”.

```

5436     \tl_set:Nn \l_@@_suffix_tl { - medium }
5437     \@@_create_nodes:
5438     \@@_computations_for_large_nodes:
5439     \tl_set:Nn \l_@@_suffix_tl { - large }
5440     \@@_create_nodes:
5441   \endpgfpicture
5442 }

```

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.

```

5443 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5444 {
5445   \int_set:Nn \l_@@_first_row_int 1
5446   \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`.

```

5447   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5448   {
5449     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5450     {
5451       (
5452         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5453         \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5454       )
5455       / 2
5456     }
5457     \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5458     { l_@@_row _ \@@_i: _ min _ dim }
5459   }
5460   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5461   {
5462     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5463     {

```

⁷²If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

5464      (
5465      \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5466      \dim_use:c
5467      { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5468      )
5469      / 2
5470      }
5471      \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5472      { l_@@_column _ \@@_j: _ max _ dim }
5473      }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

5474      \dim_sub:cn
5475      { l_@@_column _ 1 _ min _ dim }
5476      \l_@@_left_margin_dim
5477      \dim_add:cn
5478      { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5479      \l_@@_right_margin_dim
5480      }

```

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).

```

5481 \cs_new_protected:Npn \@@_create_nodes:
5482 {
5483   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5484   {
5485     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5486     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

5487       \@@_pgf_rect_node:nnnnn
5488       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5489       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
5490       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
5491       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
5492       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
5493       \str_if_empty:NF \l_@@_name_str
5494       {
5495         \pgfnodealias
5496         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5497         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5498       }
5499     }
5500   }

```

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 .

```

5501   \seq_mapthread_function:NNN
5502   \g_@@_multicolumn_cells_seq
5503   \g_@@_multicolumn_sizes_seq
5504   \@@_node_for_multicolumn:nn
5505   }

```

```

5506 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5507 {
5508   \cs_set_nopar:Npn \@@_i: { #1 }
5509   \cs_set_nopar:Npn \@@_j: { #2 }
5510 }

```

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”).

```

5511 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5512 {
5513   \@@_extract_coords_values: #1 \q_stop
5514   \@@_pgf_rect_node:nnnnn
5515     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5516     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
5517     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
5518     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
5519     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
5520   \str_if_empty:NF \l_@@_name_str
5521   {
5522     \pgfnodealias
5523       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5524       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
5525   }
5526 }

```

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.

```

5527 \keys_define:nn { NiceMatrix / Block / FirstPass }
5528 {
5529   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5530   l .value_forbidden:n = true ,
5531   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5532   r .value_forbidden:n = true ,
5533   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5534   c .value_forbidden:n = true ,
5535   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5536   L .value_forbidden:n = true ,
5537   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5538   R .value_forbidden:n = true ,
5539   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5540   C .value_forbidden:n = true ,
5541   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5542   t .value_forbidden:n = true ,
5543   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5544   b .value_forbidden:n = true ,
5545   color .tl_set:N = \l_@@_color_tl ,
5546   color .value_required:n = true
5547 }

```

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 `>`. It's mandatory to use an expandable command.

```

5548 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
5549 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not be provided by the user, you use 1-1 (that is to say a block of only one cell).

```

5550   \peek_remove_spaces:n
5551   {
5552     \tl_if_blank:nTF { #2 }

```



```

5553         { \@@_Block_i 1-1 \q_stop }
5554         { \@@_Block_i #2 \q_stop }
5555     { #1 } { #3 } { #4 }
5556 }
5557 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5558 \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.

```

5559 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5560 {

```

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).

```

5561     \bool_lazy_or:nnTF
5562     { \tl_if_blank_p:n { #1 } }
5563     { \str_if_eq_p:nn { #1 } { * } }
5564     { \int_set:Nn \l_tmpa_int { 100 } }
5565     { \int_set:Nn \l_tmpa_int { #1 } }
5566     \bool_lazy_or:nnTF
5567     { \tl_if_blank_p:n { #2 } }
5568     { \str_if_eq_p:nn { #2 } { * } }
5569     { \int_set:Nn \l_tmpb_int { 100 } }
5570     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5571     \int_compare:nNnTF \l_tmpb_int = 1
5572     {
5573         \str_if_empty:NTF \l_@@_hpos_cell_str
5574         { \str_set:Nn \l_@@_hpos_block_str c }
5575         { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
5576     }
5577     { \str_set:Nn \l_@@_hpos_block_str c }

```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```

5578     \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
5579     \tl_set:Nx \l_tmpa_tl
5580     {
5581         { \int_use:N \c@iRow }
5582         { \int_use:N \c@jCol }
5583         { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5584         { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5585     }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets: $\{imin\}\{jmin\}\{imax\}\{jmax\}$.

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by currying).

```

5586     \bool_if:nTF
5587     {
5588     (

```

```

5589     \int_compare_p:nNn { \l_tmpa_int } = 1
5590     ||
5591     \int_compare_p:nNn { \l_tmpb_int } = 1
5592   )
5593   && ! \tl_if_empty_p:n { #5 }

```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a **X** column, we should not do that since the width is determined by another way. This should be the same for the **p**, **m** and **b** columns and we should modify that point. However, for the **X** column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

5594     && ! \l_@@_X_column_bool
5595   }
5596   { \exp_args:Nxx \@@_Block_iv:nnnnn }
5597   { \exp_args:Nxx \@@_Block_v:nnnnn }
5598   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5599 }

```

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).

```

5600 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5601 {
5602   \int_gincr:N \g_@@_block_box_int
5603   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5604   {
5605     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5606     {
5607       \@@_actually_diagbox:nnnnnn
5608       { \int_use:N \c@iRow }
5609       { \int_use:N \c@jCol }
5610       { \int_eval:n { \c@iRow + #1 - 1 } }
5611       { \int_eval:n { \c@jCol + #2 - 1 } }
5612       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5613     }
5614   }
5615   \box_gclear_new:c
5616   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
5617   \hbox_gset:cn
5618   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
5619   {

```

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:` (in order to use `\color_ensure_current:` safely, you should load `l3backend` before the `\documentclass` with `\RequirePackage{expl3}`).

```

5620     \tl_if_empty:NTF \l_@@_color_tl
5621     { \int_compare:nNnT { #2 } = 1 \set@color }
5622     { \color { \l_@@_color_tl } }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even if it has yet been used in the beginning of the cell where the command `\Block` has been issued because we want to be able to take into account a potential instruction of color of the font in `\g_@@_row_style_tl`.

```

5623     \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5624     \group_begin:
5625     \cs_set:Npn \arraystretch { 1 }
5626     \dim_zero:N \extrarowheight
5627     #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).

```

5628 \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
5629 \bool_if:NTF \l_@@_NiceTabular_bool
5630 {
5631   \bool_lazy_and:nnTF
5632   { \int_compare_p:nNn { #2 } = 1 }
5633   { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }

```

When the block is mono-column in a column with a fixed width (eg p{3cm}).

```

5634 {
5635   \begin { minipage } [ \l_@@_vpos_of_block_tl ]
5636   { \l_@@_col_width_dim }
5637   \str_case:Vn \l_@@_hpos_block_str
5638   {
5639     c \centering
5640     r \raggedleft
5641     l \raggedright
5642   }
5643   #5
5644   \end { minipage }
5645 }
5646 {
5647   \use:x
5648   {
5649     \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5650     { @ { } \l_@@_hpos_block_str @ { } }
5651   }
5652   #5
5653   \end { tabular }
5654 }
5655 {
5656   \c_math_toggle_token
5657   \use:x
5658   {
5659     \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5660     { @ { } \l_@@_hpos_block_str @ { } }
5661   }
5662   #5
5663   \end { array }
5664   \c_math_toggle_token
5665 }
5666 \group_end:
5667 }
5668 \bool_if:NT \g_@@_rotate_bool
5669 {
5670   \box_grotate:cn
5671   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5672   { 90 }
5673   \bool_gset_false:N \g_@@_rotate_bool
5674 }
5675 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5676 \int_compare:nNnT { #2 } = 1
5677 {
5678   \dim_gset:Nn \g_@@_blocks_wd_dim
5679   {
5680     \dim_max:nn
5681     \g_@@_blocks_wd_dim
5682     {
5683       \box_wd:c

```

```

5684         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5685     }
5686 }
5687 }

```

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.

```

5688 \int_compare:nNnT { #1 } = 1
5689 {
5690     \dim_gset:Nn \g_@@_blocks_ht_dim
5691     {
5692         \dim_max:nn
5693         \g_@@_blocks_ht_dim
5694         {
5695             \box_ht:c
5696             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5697         }
5698     }
5699     \dim_gset:Nn \g_@@_blocks_dp_dim
5700     {
5701         \dim_max:nn
5702         \g_@@_blocks_dp_dim
5703         {
5704             \box_dp:c
5705             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5706         }
5707     }
5708 }
5709 \seq_gput_right:Nx \g_@@_blocks_seq
5710 {
5711     \l_tmpa_tl

```

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in \l_@@_hpos_block_str. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of \l_@@_hpos_block_str, which is fixed by the type of current column.

```

5712     { \exp_not:n { #3 } , \l_@@_hpos_block_str }
5713     {
5714         \box_use_drop:c
5715         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5716     }
5717 }
5718 }

```

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.

```

5719 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5720 {
5721     \seq_gput_right:Nx \g_@@_blocks_seq
5722     {
5723         \l_tmpa_tl
5724         { \exp_not:n { #3 } }
5725         \exp_not:n
5726         {
5727             {
5728                 \bool_if:NTF \l_@@_NiceTabular_bool
5729                 {
5730                     \group_begin:
5731                     \cs_set:Npn \arraystretch { 1 }
5732                     \dim_zero:N \extrarowheight
5733                     #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).

```

5734         \bool_if:NT \g_@@_rotate_bool
5735         { \str_set:Nn \l_@@_hpos_block_str c }
5736     \use:x
5737     {
5738         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5739         { @ { } \l_@@_hpos_block_str @ { } }
5740     }
5741     #5
5742     \end { tabular }
5743     \group_end:
5744 }
5745 {
5746     \group_begin:
5747     \cs_set:Npn \arraystretch { 1 }
5748     \dim_zero:N \extrarowheight
5749     #4
5750     \bool_if:NT \g_@@_rotate_bool
5751     { \str_set:Nn \l_@@_hpos_block_str c }
5752     \c_math_toggle_token
5753     \use:x
5754     {
5755         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5756         { @ { } \l_@@_hpos_block_str @ { } }
5757     }
5758     #5
5759     \end { array }
5760     \c_math_toggle_token
5761     \group_end:
5762 }
5763 }
5764 }
5765 }
5766 }
```

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).

```

5767 \keys_define:nn { NiceMatrix / Block / SecondPass }
5768 {
5769     tikz .code:n =
5770         \bool_if:NTF \c_@@_tikz_loaded_bool
5771         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
5772         { \@@_error:n { tikz-key-without~tikz } } ,
5773     tikz .value_required:n = true ,
5774     fill .tl_set:N = \l_@@_fill_tl ,
5775     fill .value_required:n = true ,
5776     draw .tl_set:N = \l_@@_draw_tl ,
5777     draw .default:n = default ,
5778     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5779     rounded-corners .default:n = 4 pt ,
5780     color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5781     color .value_required:n = true ,
5782     borders .clist_set:N = \l_@@_borders_clist ,
5783     borders .value_required:n = true ,
5784     hvlines .bool_set:N = \l_@@_hvlines_block_bool ,
5785     hvlines .default:n = true ,
5786     line-width .dim_set:N = \l_@@_line_width_dim ,
```

```

5787   line-width .value_required:n = true ,
5788   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5789   l .value_forbidden:n = true ,
5790   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5791   r .value_forbidden:n = true ,
5792   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5793   c .value_forbidden:n = true ,
5794   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
5795           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5796   L .value_forbidden:n = true ,
5797   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
5798           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5799   R .value_forbidden:n = true ,
5800   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
5801           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5802   C .value_forbidden:n = true ,
5803   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5804   t .value_forbidden:n = true ,
5805   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5806   b .value_forbidden:n = true ,
5807   name .tl_set:N = \l_@@_block_name_str ,
5808   name .value_required:n = true ,
5809   name .initial:n = \c_empty_tl ,
5810   unknown .code:n = \@@_error:n { Unknown-key-for-Block }
5811 }

```

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.

```

5812 \cs_new_protected:Npn \@@_draw_blocks:
5813 {
5814   \cs_set_eq:NN \ialign \@@_old_ialign:
5815   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
5816 }
5817 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5818 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5819   \int_zero_new:N \l_@@_last_row_int
5820   \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.

```

5821   \int_compare:nNnTF { #3 } > { 99 }
5822   { \int_set_eq:NN \l_@@_last_row_int \c_iRow }
5823   { \int_set:Nn \l_@@_last_row_int { #3 } }
5824   \int_compare:nNnTF { #4 } > { 99 }
5825   { \int_set_eq:NN \l_@@_last_col_int \c_jCol }
5826   { \int_set:Nn \l_@@_last_col_int { #4 } }
5827   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
5828   {
5829     \int_compare:nTF
5830     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
5831     {
5832       \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
5833       \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
5834       \group_begin:

```

```

5835         \globaldefs = 1
5836         \@@_msg_redirect_name:nn { columns~not~used } { none }
5837         \group_end:
5838     }
5839     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5840 }
5841 {
5842     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
5843     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5844     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
5845 }
5846 }
5847 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
5848 {

```

The group is for the keys.

```

5849     \group_begin:
5850     \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
5851     \bool_if:NTF \l_@@_hvlines_block_bool
5852     {
5853         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5854         {
5855             \@@_hvlines_block:nnn
5856             { \exp_not:n { #5 } }
5857             { #1 - #2 }
5858             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5859         }
5860     }
5861     {

```

The sequence of the positions of the blocks (excepted the blocks with the key hvlines) will be used when drawing the rules (in fact, there is also the \multicolumn and the \diagbox in that sequence).

```

5862         \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
5863         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
5864     }
5865     \tl_if_empty:NF \l_@@_draw_tl
5866     {
5867         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5868         {
5869             \@@_stroke_block:nnn
5870             { \exp_not:n { #5 } }
5871             { #1 - #2 }
5872             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5873         }
5874         \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
5875         { { #1 } { #2 } { #3 } { #4 } }
5876     }
5877     \clist_if_empty:NF \l_@@_borders_clist
5878     {
5879         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5880         {
5881             \@@_stroke_borders_block:nnn
5882             { \exp_not:n { #5 } }
5883             { #1 - #2 }
5884             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5885         }
5886     }
5887     \tl_if_empty:NF \l_@@_fill_tl
5888     {

```

The command \@@_extract_brackets will extract the potential specification of color space at the beginning of \l_@@_fill_tl and store it in \l_tmpa_tl and store the color itself in \l_tmpb_tl.

```

5889 \exp_last_unbraced:N \l_@@_extract_brackets \l_@@_fill_tl \q_stop
5890 \tl_gput_right:Nx \g_nicematrix_code_before_tl
5891 {
5892   \exp_not:N \roundedrectanglecolor
5893   [ \l_tmpa_tl ]
5894   { \exp_not:N \l_tmpb_tl }
5895   { #1 - #2 }
5896   { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5897   { \dim_use:N \l_@@_rounded_corners_dim }
5898 }
5899 }
5900 \seq_if_empty:NF \l_@@_tikz_seq
5901 {
5902   \tl_gput_right:Nx \g_nicematrix_code_before_tl
5903   {
5904     \@@_block_tikz:nnnnn
5905     { #1 }
5906     { #2 }
5907     { \int_use:N \l_@@_last_row_int }
5908     { \int_use:N \l_@@_last_col_int }
5909     { \seq_use:Nn \l_@@_tikz_seq { , } }
5910   }
5911 }
5912 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5913 {
5914   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5915   {
5916     \@@_actually_diagbox:nnnnnn
5917     { #1 }
5918     { #2 }
5919     { \int_use:N \l_@@_last_row_int }
5920     { \int_use:N \l_@@_last_col_int }
5921     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5922   }
5923 }
5924 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
5925 \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`.

```

\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.

```

5926 \pgfpicture

```



```

5927 \pgfrememberpicturepositiononpagetrue
5928 \pgf@relevantforpicturesizefalse
5929 \@@_qpoint:n { row - #1 }
5930 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5931 \@@_qpoint:n { col - #2 }
5932 \dim_set_eq:NN \l_tmpb_dim \pgf@x
5933 \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
5934 \dim_set_eq:NN \l_tmpc_dim \pgf@y
5935 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5936 \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.

```

5937 \@@_pgf_rect_node:nnnnn
5938 { \@@_env: - #1 - #2 - block }
5939 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
5940 \str_if_empty:NF \l_@@_block_name_str
5941 {
5942   \pgfnodealias
5943   { \@@_env: - \l_@@_block_name_str }
5944   { \@@_env: - #1 - #2 - block }
5945   \str_if_empty:NF \l_@@_name_str
5946   {
5947     \pgfnodealias
5948     { \l_@@_name_str - \l_@@_block_name_str }
5949     { \@@_env: - #1 - #2 - block }
5950   }
5951 }

```

Now, we create the “short node” which, in general, will be used to put the label (that is to say the content of the node). However, if one the keys L, C or R is used (that information is provided by the boolean `\l_@@_hpos_of_block_cap_bool`), we don’t need to create that node since the normal node is used to put the label.

```

5952 \bool_if:NF \l_@@_hpos_of_block_cap_bool
5953 {
5954   \dim_set_eq:NN \l_tmpb_dim \c_max_dim

```

The short node is constructed by taking into account the *contents* of the columns involved in at least one cell of the block. That’s why we have to do a loop over the rows of the array.

```

5955 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5956 {

```

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.

```

5957 \cs_if_exist:cT
5958 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5959 {
5960   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5961   {
5962     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
5963     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
5964   }
5965 }
5966 }

```

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.

```

5967 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
5968 {
5969   \@@_qpoint:n { col - #2 }
5970   \dim_set_eq:NN \l_tmpb_dim \pgf@x
5971 }
5972 \dim_set:Nn \l_tmpd_dim { - \c_max_dim }

```

```

5973 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5974 {
5975   \cs_if_exist:cT
5976   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5977   {
5978     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5979     {
5980       \pgfpointanchor
5981       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5982       { east }
5983       \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
5984     }
5985   }
5986 }
5987 \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
5988 {
5989   \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5990   \dim_set_eq:NN \l_tmpd_dim \pgf@x
5991 }
5992 \@@_pgf_rect_node:nnnnn
5993 { \@@_env: - #1 - #2 - block - short }
5994 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
5995 }

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@_pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```

5996 \bool_if:NT \l_@@_medium_nodes_bool
5997 {
5998   \@@_pgf_rect_node:nnn
5999   { \@@_env: - #1 - #2 - block - medium }
6000   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
6001   {
6002     \pgfpointanchor
6003     { \@@_env:
6004       - \int_use:N \l_@@_last_row_int
6005       - \int_use:N \l_@@_last_col_int - medium
6006     }
6007     { south-east }
6008   }
6009 }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

6010 \int_compare:nNnTF { #1 } = { #3 }
6011 {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

6012 \int_compare:nNnTF { #1 } = 0
6013 { \l_@@_code_for_first_row_tl }
6014 {
6015   \int_compare:nNnT { #1 } = \l_@@_last_row_int
6016   \l_@@_code_for_last_row_tl
6017 }

```

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`.

```

6018 \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

6019 \pgfpointanchor
6020 {
6021   \@@_env: - #1 - #2 - block
6022   \bool_if:Nf \l_@@_hpos_of_block_cap_bool { - short }
6023 }

```

```

6024     {
6025         \str_case:Vn \l_@@_hpos_block_str
6026         {
6027             c { center }
6028             l { west }
6029             r { east }
6030         }
6031     }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

6032     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
6033     \pgfset { inner-sep = \c_zero_dim }
6034     \pgfnode
6035     { rectangle }
6036     {
6037         \str_case:Vn \l_@@_hpos_block_str
6038         {
6039             c { base }
6040             l { base~west }
6041             r { base~east }
6042         }
6043     }
6044     { \box_use_drop:N \l_@@_cell_box } { } { }
6045 }

```

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).

```

6046     {

```

If we are in the first column, we must put the block as if it was with the key r.

```

6047     \int_compare:nNnT { #2 } = 0
6048     { \str_set:Nn \l_@@_hpos_block_str r }
6049     \bool_if:nT \g_@@_last_col_found_bool
6050     {
6051         \int_compare:nNnT { #2 } = \g_@@_col_total_int
6052         { \str_set:Nn \l_@@_hpos_block_str l }
6053     }
6054     \pgftransformshift
6055     {
6056         \pgfpointanchor
6057         {
6058             \@@_env: - #1 - #2 - block
6059             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6060         }
6061         {
6062             \str_case:Vn \l_@@_hpos_block_str
6063             {
6064                 c { center }
6065                 l { west }
6066                 r { east }
6067             }
6068         }
6069     }
6070     \pgfset { inner-sep = \c_zero_dim }
6071     \pgfnode
6072     { rectangle }
6073     {
6074         \str_case:Vn \l_@@_hpos_block_str
6075         {
6076             c { center }
6077             l { west }
6078             r { east }
6079         }
6080     }

```

```

6081     { \box_use_drop:N \l_@@_cell_box } { } { }
6082   }
6083   \endpgfpicture
6084   \group_end:
6085 }

6086 \NewDocumentCommand \@@_extract_brackets { 0 { } }
6087 {
6088   \tl_set:Nn \l_tmpa_tl { #1 }
6089   \@@_store_in_tmpb_tl
6090 }
6091 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
6092 { \tl_set:Nn \l_tmpb_tl { #1 } }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

6093 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
6094 {
6095   \group_begin:
6096   \tl_clear:N \l_@@_draw_tl
6097   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6098   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
6099   \pgfpicture
6100   \pgfrememberpicturepositiononpagetrue
6101   \pgf@relevantforpicturesizefalse
6102   \tl_if_empty:NF \l_@@_draw_tl
6103   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

6104     \str_if_eq:VnTF \l_@@_draw_tl { default }
6105     { \CT@arc@ }
6106     { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
6107   }
6108   \pgfsetcornersarced
6109   {
6110     \pgfpoint
6111     { \dim_use:N \l_@@_rounded_corners_dim }
6112     { \dim_use:N \l_@@_rounded_corners_dim }
6113   }
6114   \@@_cut_on_hyphen:w #2 \q_stop
6115   \bool_lazy_and:nnT
6116   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
6117   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
6118   {
6119     \@@_qpoint:n { row - \l_tmpa_tl }
6120     \dim_set:Nn \l_tmpb_dim { \pgf@y }
6121     \@@_qpoint:n { col - \l_tmpb_tl }
6122     \dim_set:Nn \l_tmpc_dim { \pgf@x }
6123     \@@_cut_on_hyphen:w #3 \q_stop
6124     \int_compare:nNnT \l_tmpa_tl > \c@iRow
6125     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
6126     \int_compare:nNnT \l_tmpb_tl > \c@jCol
6127     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
6128     \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
6129     \dim_set:Nn \l_tmpa_dim { \pgf@y }
6130     \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
6131     \dim_set:Nn \l_tmpd_dim { \pgf@x }
6132     \pgfpathrectanglecorners
6133     { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
6134     { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
6135     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

6136     \pgfusepath { stroke }
6137   }
6138   \endpgfpicture
6139   \group_end:
6140 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

6141 \keys_define:nn { NiceMatrix / BlockStroke }
6142 {
6143   color .tl_set:N = \l_@@_draw_tl ,
6144   draw .tl_set:N = \l_@@_draw_tl ,
6145   draw .default:n = default ,
6146   line-width .dim_set:N = \l_@@_line_width_dim ,
6147   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6148   rounded-corners .default:n = 4 pt
6149 }

```

The first argument of `\@@_hvlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

6150 \cs_new_protected:Npn \@@_hvlines_block:nnn #1 #2 #3
6151 {
6152   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6153   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6154   \@@_cut_on_hyphen:w #2 \q_stop
6155   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6156   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6157   \@@_cut_on_hyphen:w #3 \q_stop
6158   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6159   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6160   \int_step_inline:nnn \l_tmpd_tl \l_tmpb_tl
6161   {
6162     \use:x
6163     { \@@_vline:nnnn { ##1 } 1 { \l_tmpc_tl } { \@@_pred:n \l_tmpa_tl } }
6164   }
6165   \int_step_inline:nnn \l_tmpc_tl \l_tmpa_tl
6166   {
6167     \use:x
6168     { \@@_hline:nnnn { ##1 } 1 { \l_tmpd_tl } { \@@_pred:n \l_tmpb_tl } }
6169   }
6170 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

6171 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
6172 {
6173   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6174   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6175   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
6176   { \@@_error:n { borders~forbidden } }
6177   {
6178     \clist_map_inline:Nn \l_@@_borders_clist
6179     {
6180       \clist_if_in:nnF { top , bottom , left , right } { ##1 }
6181       { \@@_error:nn { bad~border } { ##1 } }
6182     }
6183     \@@_cut_on_hyphen:w #2 \q_stop
6184     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6185     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6186     \@@_cut_on_hyphen:w #3 \q_stop

```

```

6187 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6188 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6189 \pgfpicture
6190 \pgfrememberpicturepositiononpagetrue
6191 \pgf@relevantforpicturesizefalse
6192 \CT@arc@
6193 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
6194 \clist_if_in:NnT \l_@@_borders_clist { right }
6195 { \@@_stroke_vertical:n \l_tmpb_tl }
6196 \clist_if_in:NnT \l_@@_borders_clist { left }
6197 { \@@_stroke_vertical:n \l_tmpd_tl }
6198 \clist_if_in:NnT \l_@@_borders_clist { bottom }
6199 { \@@_stroke_horizontal:n \l_tmpa_tl }
6200 \clist_if_in:NnT \l_@@_borders_clist { top }
6201 { \@@_stroke_horizontal:n \l_tmpc_tl }
6202 \endpgfpicture
6203 }
6204 }

```

The following command is used to stroke the left border and the right border. The argument #1 is the number of column (in the sense of the col node).

```

6205 \cs_new_protected:Npn \@@_stroke_vertical:n #1
6206 {
6207   \@@_qpoint:n \l_tmpc_tl
6208   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6209   \@@_qpoint:n \l_tmpa_tl
6210   \dim_set:Nn \l_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6211   \@@_qpoint:n { #1 }
6212   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
6213   \pgfpathlineto { \pgfpoint \pgf@x \l_tmpc_dim }
6214   \pgfusepathqstroke
6215 }

```

The following command is used to stroke the top border and the bottom border. The argument #1 is the number of row (in the sense of the row node).

```

6216 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
6217 {
6218   \@@_qpoint:n \l_tmpd_tl
6219   \clist_if_in:NnTF \l_@@_borders_clist { left }
6220   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
6221   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
6222   \@@_qpoint:n \l_tmpb_tl
6223   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
6224   \@@_qpoint:n { #1 }
6225   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
6226   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6227   \pgfusepathqstroke
6228 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

6229 \keys_define:nn { NiceMatrix / BlockBorders }
6230 {
6231   borders .clist_set:N = \l_@@_borders_clist ,
6232   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6233   rounded-corners .default:n = 4 pt ,
6234   line-width .dim_set:N = \l_@@_line_width_dim
6235 }

```

The following command will be used if the key tikz has been used for the command \Block. The arguments #1 and #2 are the coordinates of the first cell and #3 and #4 the coordinates of the last cell of the block. #5 is a comma-separated list of the Tikz keys used with the path.

```

6236 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5

```

```

6237 {
6238   \begin { tikzpicture }
6239   \clist_map_inline:nn { #5 }
6240     {
6241       \path [ ##1 ]
6242         ( #1 -| #2 ) rectangle ( \@@_succ:n { #3 } -| \@@_succ:n { #4 } ) ;
6243     }
6244   \end { tikzpicture }
6245 }

```

How to draw the dotted lines transparently

```

6246 \cs_set_protected:Npn \@@_renew_matrix:
6247 {
6248   \RenewDocumentEnvironment { pmatrix } { } {
6249     { \pNiceMatrix }
6250     { \endpNiceMatrix }
6251   \RenewDocumentEnvironment { vmatrix } { } {
6252     { \vNiceMatrix }
6253     { \endvNiceMatrix }
6254   \RenewDocumentEnvironment { Vmatrix } { } {
6255     { \VNiceMatrix }
6256     { \endVNiceMatrix }
6257   \RenewDocumentEnvironment { bmatrix } { } {
6258     { \bNiceMatrix }
6259     { \endbNiceMatrix }
6260   \RenewDocumentEnvironment { Bmatrix } { } {
6261     { \BNiceMatrix }
6262     { \endBNiceMatrix }
6263 }

```

Automatic arrays

```

6264 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
6265 {
6266   \int_set:Nn \l_@@_nb_rows_int { #1 }
6267   \int_set:Nn \l_@@_nb_cols_int { #2 }
6268 }

```

We will extract the potential keys l, r and c and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

6269 \keys_define:nn { NiceMatrix / Auto }
6270 {
6271   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
6272   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
6273   c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
6274 }
6275 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
6276 {
6277   \int_zero_new:N \l_@@_nb_rows_int
6278   \int_zero_new:N \l_@@_nb_cols_int
6279   \@@_set_size:n #4 \q_stop

```

The group is for the protection of `\l_@@_type_of_col_tl`.

```

6280   \group_begin:
6281   \tl_set:Nn \l_@@_type_of_col_tl c
6282   \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
6283   \use:x
6284   {
6285     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
6286     { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
6287     [ \exp_not:N \l_tmpa_tl ]

```

```

6288     }
6289     \int_compare:nNnT \l_@@_first_row_int = 0
6290     {
6291         \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6292         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { & }
6293         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6294     }
6295     \prg_replicate:nn \l_@@_nb_rows_int
6296     {
6297         \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

We 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).

```

6298         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
6299         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6300     }
6301     \int_compare:nNnT \l_@@_last_row_int > { -2 }
6302     {
6303         \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6304         \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6305         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6306     }
6307     \end { NiceArrayWithDelims }
6308     \group_end:
6309 }
6310 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6311 {
6312     \cs_set_protected:cpn { #1 AutoNiceMatrix }
6313     {
6314         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6315         \AutoNiceMatrixWithDelims { #2 } { #3 }
6316     }
6317 }
6318 \@@_define_com:nnn p ( )
6319 \@@_define_com:nnn b [ ]
6320 \@@_define_com:nnn v | |
6321 \@@_define_com:nnn V \ | \ |
6322 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

6323 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
6324 {
6325     \group_begin:
6326     \bool_set_true:N \l_@@_NiceArray_bool
6327     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6328     \group_end:
6329 }

```

The redefinition of the command \dotfill

```

6330 \cs_set_eq:NN \@@_old_dotfill \dotfill
6331 \cs_new_protected:Npn \@@_dotfill:
6332 {

```

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}).

```

6333     \@@_old_dotfill
6334     \bool_if:NT \l_@@_NiceTabular_bool
6335     { \group_insert_after:N \@@_dotfill_ii: }
6336     { \group_insert_after:N \@@_dotfill_i: }
6337 }
6338 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }

```



```
6339 \cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }
```

Now, if the box is not empty (unfortunately, we can't actually test whether the box is empty and that's why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```
6340 \cs_new_protected:Npn \@@_dotfill_iii:
6341 { \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`. However, there are also redefinitions of `\diagbox` in other circumstances.

```
6342 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6343 {
6344   \tl_gput_right:Nx \g_@@_internal_code_after_tl
6345   {
6346     \@@_actually_diagbox:nnnnnn
6347     { \int_use:N \c@iRow }
6348     { \int_use:N \c@jCol }
6349     { \int_use:N \c@iRow }
6350     { \int_use:N \c@jCol }
6351     { \exp_not:n { #1 } }
6352     { \exp_not:n { #2 } }
6353   }
```

We put the cell with `\diagbox` in the sequence `\g_@@_pos_of_blocks_seq` because a cell with `\diagbox` must be considered as non empty by the key `corners`.

```
6354   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6355   {
6356     { \int_use:N \c@iRow }
6357     { \int_use:N \c@jCol }
6358     { \int_use:N \c@iRow }
6359     { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
6360     { }
6361   }
6362 }
```

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 other two are the elements to draw below and above the diagonal line.

```
6363 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
6364 {
6365   \pgfpicture
6366   \pgf@relevantforpicturesizefalse
6367   \pgfrememberpicturepositiononpagetrue
6368   \@@_qpoint:n { row - #1 }
6369   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6370   \@@_qpoint:n { col - #2 }
6371   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6372   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6373   \@@_qpoint:n { row - \@@_succ:n { #3 } }
6374   \dim_set_eq:NN \l_tmpc_dim \pgf@y
6375   \@@_qpoint:n { col - \@@_succ:n { #4 } }
6376   \dim_set_eq:NN \l_tmpd_dim \pgf@x
6377   \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
6378   {
```

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.

```

6379     \CT@arc@
6380     \pgfsetroundcap
6381     \pgfusepathqstroke
6382 }
6383 \pgfset { inner~sep = 1 pt }
6384 \pgfscope
6385 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
6386 \pgfnode { rectangle } { south-west }
6387 {
6388     \begin { minipage } { 20 cm }
6389     \@@_math_toggle_token: #5 \@@_math_toggle_token:
6390     \end { minipage }
6391 }
6392 { }
6393 { }
6394 \endpgfscope
6395 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
6396 \pgfnode { rectangle } { north-east }
6397 {
6398     \begin { minipage } { 20 cm }
6399     \raggedleft
6400     \@@_math_toggle_token: #6 \@@_math_toggle_token:
6401     \end { minipage }
6402 }
6403 { }
6404 { }
6405 \endpgfpicture
6406 }
```

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.

```

6407 \keys_define:nn { NiceMatrix }
6408 {
6409     CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6410     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6411 }
6412 \keys_define:nn { NiceMatrix / CodeAfter }
6413 {
6414     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6415     sub-matrix .value_required:n = true ,
6416     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6417     delimiters / color .value_required:n = true ,
6418     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6419     rules .value_required:n = true ,
6420     unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
6421 }
```

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. 120.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

6422 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_ii:n }
```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_ii:n` which begins with `\`.

```
6423 \cs_new_protected:Npn \@@_CodeAfter_i: { \ \omit \@@_CodeAfter_ii:n }
```

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```
6424 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
6425 {
6426   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6427   \@@_CodeAfter_iv:n
6428 }
```

We catch the argument of the command `\end` (in `#1`).

```
6429 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
6430 {
```

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.

```
6431   \str_if_eq:eeTF \@currenvir { #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`.

```
6432   {
6433     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6434     \@@_CodeAfter_ii:n
6435   }
6436 }
```

The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_internal_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of 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.

```
6437 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
6438 {
6439   \pgfpicture
6440   \pgfrememberpicturepositiononpagetrue
6441   \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.

```
6442   \@@_qpoint:n { row - 1 }
6443   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6444   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
6445   \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).

```
6446   \bool_if:nTF { #3 }
6447   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6448   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6449   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6450   {
6451     \cs_if_exist:cT
```

```

6452 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6453 {
6454   \pgfpointanchor
6455   { \@@_env: - ##1 - #2 }
6456   { \bool_if:nTF { #3 } { west } { east } }
6457   \dim_set:Nn \l_tmpa_dim
6458   { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6459 }
6460 }

```

Now we can put the delimiter with a node of PGF.

```

6461 \pgfset { inner~sep = \c_zero_dim }
6462 \dim_zero:N \nulldelimiterspace
6463 \pgftransformshift
6464 {
6465   \pgfpoint
6466   { \l_tmpa_dim
6467     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6468   }
6469 \pgfnode
6470 { rectangle }
6471 { \bool_if:nTF { #3 } { east } { west } }
6472 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

6473 \nullfont
6474 \c_math_toggle_token
6475 \tl_if_empty:NF \l_@@_delimiters_color_tl
6476 { \color { \l_@@_delimiters_color_tl } }
6477 \bool_if:nTF { #3 } { \left #1 } { \left . }
6478 \vcenter
6479 {
6480   \nullfont
6481   \hrule \@height
6482     \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6483     \@depth \c_zero_dim
6484     \@width \c_zero_dim
6485   }
6486   \bool_if:nTF { #3 } { \right . } { \right #1 }
6487   \c_math_toggle_token
6488 }
6489 { }
6490 { }
6491 \endpgfpicture
6492 }

```

The command `\SubMatrix`

```

6493 \keys_define:nn { NiceMatrix / sub-matrix }
6494 {
6495   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6496   extra-height .value_required:n = true ,
6497   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6498   left-xshift .value_required:n = true ,
6499   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6500   right-xshift .value_required:n = true ,
6501   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6502   xshift .value_required:n = true ,
6503   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6504   delimiters / color .value_required:n = true ,
6505   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6506   slim .default:n = true ,
6507   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6508   hlines .default:n = all ,

```

```

6509   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6510   vlines .default:n = all ,
6511   hvlines .meta:n = { hlines, vlines } ,
6512   hvlines .value_forbidden:n = true ,
6513 }
6514 \keys_define:nn { NiceMatrix }
6515 {
6516   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6517   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6518   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6519   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6520   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6521   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6522 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

6523 \keys_define:nn { NiceMatrix / SubMatrix }
6524 {
6525   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6526   hlines .default:n = all ,
6527   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6528   vlines .default:n = all ,
6529   hvlines .meta:n = { hlines, vlines } ,
6530   hvlines .value_forbidden:n = true ,
6531   name .code:n =
6532     \tl_if_empty:nTF { #1 }
6533     { \@@_error:n { Invalid-name-format } }
6534     {
6535       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6536       {
6537         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6538         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
6539         {
6540           \str_set:Nn \l_@@_submatrix_name_str { #1 }
6541           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6542         }
6543       }
6544       { \@@_error:n { Invalid-name-format } }
6545     } ,
6546   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6547   rules .value_required:n = true ,
6548   code .tl_set:N = \l_@@_code_tl ,
6549   code .value_required:n = true ,
6550   name .value_required:n = true ,
6551   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
6552 }

6553 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
6554 {
6555   \peek_remove_spaces:n
6556   {
6557     \@@_cut_on_hyphen:w #3 \q_stop
6558     \tl_clear_new:N \l_tmpc_tl
6559     \tl_clear_new:N \l_tmpd_tl
6560     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6561     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6562     \@@_cut_on_hyphen:w #2 \q_stop
6563     \seq_gput_right:Nx \g_@@_submatrix_seq
6564     { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
6565     \tl_gput_right:Nn \g_@@_internal_code_after_tl
6566     { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6567   }

```

6568 }

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;
- #6 is the potential subscript;
- #7 is the potential superscript.

For explanations about the construction with rescanning of the preamble, see the documentation for the user command `\Cdots`.

```

6569 \AtBeginDocument
6570 {
6571   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
6572   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
6573   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
6574     {
6575       \peek_remove_spaces:n
6576       {
6577         \@@_sub_matrix:nnnnnnn
6578         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
6579       }
6580     }
6581 }

6582 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6583 {
6584   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

6585   \tl_clear_new:N \l_@@_first_i_tl
6586   \tl_clear_new:N \l_@@_first_j_tl
6587   \tl_clear_new:N \l_@@_last_i_tl
6588   \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`.

```

6589   \@@_cut_on_hyphen:w #2 \q_stop
6590   \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
6591   \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
6592   \@@_cut_on_hyphen:w #3 \q_stop
6593   \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
6594   \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
6595   \bool_lazy_or:nnTF
6596     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6597     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6598     { \@@_error:n { SubMatrix~too~large } }
6599     {
6600       \str_clear_new:N \l_@@_submatrix_name_str
6601       \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6602       \pgfpicture
6603       \pgfrememberpicturerepositiononpagetrue
6604       \pgf@relevantforpicturesizefalse
6605       \pgfset { inner~sep = \c_zero_dim }
6606       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6607       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currification.

```

6608     \bool_if:NTF \l_@@_submatrix_slim_bool
6609     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6610     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6611     {
6612         \cs_if_exist:cT
6613         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6614         {
6615             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6616             \dim_set:Nn \l_@@_x_initial_dim
6617             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6618         }
6619         \cs_if_exist:cT
6620         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6621         {
6622             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6623             \dim_set:Nn \l_@@_x_final_dim
6624             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6625         }
6626     }
6627     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
6628     { \@@_error:nn { impossible~delimiter } { left } }
6629     {
6630         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
6631         { \@@_error:nn { impossible~delimiter } { right } }
6632         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
6633     }
6634     \endpgfpicture
6635 }
6636 \group_end:
6637 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

6638 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
6639 {
6640     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
6641     \dim_set:Nn \l_@@_y_initial_dim
6642     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
6643     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
6644     \dim_set:Nn \l_@@_y_final_dim
6645     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
6646     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
6647     {
6648         \cs_if_exist:cT
6649         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
6650         {
6651             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
6652             \dim_set:Nn \l_@@_y_initial_dim
6653             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6654         }
6655         \cs_if_exist:cT
6656         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6657         {
6658             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6659             \dim_set:Nn \l_@@_y_final_dim
6660             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6661         }
6662     }
6663     \dim_set:Nn \l_tmpa_dim
6664     {
6665         \l_@@_y_initial_dim - \l_@@_y_final_dim +
6666         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6667     }

```

```
6668 \dim_zero:N \nulldelimiterspace
```

We will draw the rules in the `\SubMatrix`.

```
6669 \group_begin:
6670 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6671 \tl_if_empty:NF \l_@@_rules_color_tl
6672 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6673 \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`.

```
6674 \seq_map_inline:Nn \g_@@_cols_vlism_seq
6675 {
6676   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6677   {
6678     \int_compare:nNnT
6679       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6680     {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
6681       \@@_qpoint:n { col - ##1 }
6682       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6683       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6684       \pgfusepathqstroke
6685     }
6686   }
6687 }
```

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.

```
6688 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6689 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6690 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6691 {
6692   \bool_lazy_and:nnTF
6693   { \int_compare_p:nNn { ##1 } > 0 }
6694   {
6695     \int_compare_p:nNn
6696     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6697   {
6698     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6699     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6700     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6701     \pgfusepathqstroke
6702   }
6703   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
6704 }
```

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.

```
6705 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6706 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6707 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
6708 {
6709   \bool_lazy_and:nnTF
6710   { \int_compare_p:nNn { ##1 } > 0 }
6711   {
6712     \int_compare_p:nNn
6713     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6714   {
6715     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
```


We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

6716         \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
6717         \dim_set:Nn \l_tmpa_dim
6718         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6719         \str_case:nn { #1 }
6720         {
6721             ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6722             [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
6723             \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6724             }
6725         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

6726         \dim_set:Nn \l_tmpb_dim
6727         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6728         \str_case:nn { #2 }
6729         {
6730             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6731             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
6732             \{ { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6733             }
6734         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6735         \pgfusepathqstroke
6736         \group_end:
6737     }
6738     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
6739 }

```

If the key name has been used for the command `\SubMatrix`, we create a PGF node with that name for the submatrix (this node does not encompass the delimiters that we will put after).

```

6740     \str_if_empty:NF \l_@@_submatrix_name_str
6741     {
6742         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
6743         \l_@@_x_initial_dim \l_@@_y_initial_dim
6744         \l_@@_x_final_dim \l_@@_y_final_dim
6745     }
6746     \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`.

```

6747     \begin { pgfscope }
6748     \pgftransformshift
6749     {
6750         \pgfpoint
6751         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6752         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6753     }
6754     \str_if_empty:NTF \l_@@_submatrix_name_str
6755     { \@@_node_left:nn #1 { } }
6756     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
6757     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

6758     \pgftransformshift
6759     {
6760         \pgfpoint
6761         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6762         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6763     }
6764     \str_if_empty:NTF \l_@@_submatrix_name_str

```

```

6765 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
6766 {
6767   \@@_node_right:nnnn #2
6768   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
6769 }
6770 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
6771 \flag_clear_new:n { nicematrix }
6772 \l_@@_code_tl
6773 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

6774 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

6775 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
6776 {
6777   \use:e
6778   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
6779 }

```

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 -.

```

6780 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
6781 { #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.

```

6782 \tl_const:Nn \c_@@_integers_alist_tl
6783 {
6784   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
6785   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
6786   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
6787   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
6788 }

```

```

6789 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
6790 {

```

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`.

```

6791   \tl_if_empty:nTF { #2 }
6792   {
6793     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
6794     {
6795       \flag_raise:n { nicematrix }
6796       \int_if_even:nTF { \flag_height:n { nicematrix } }
6797       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }

```

```

6798         { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
6799     }
6800     { #1 }
6801 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

6802     { \@@_pgfpointanchor_iii:w { #1 } #2 }
6803 }

```

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`).

```

6804 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
6805 {
6806     \str_case:nnF { #1 }
6807     {
6808         { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
6809         { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
6810     }

```

Now the case of a node of the form $i-j$.

```

6811     {
6812         \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
6813         - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
6814     }
6815 }

```

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`).

```

6816 \cs_new_protected:Npn \@@_node_left:nn #1 #2
6817 {
6818     \pgfnode
6819     { rectangle }
6820     { east }
6821     {
6822         \nullfont
6823         \c_math_toggle_token
6824         \tl_if_empty:NF \l_@@_delimiters_color_tl
6825         { \color { \l_@@_delimiters_color_tl } }
6826         \left #1
6827         \vcenter
6828         {
6829             \nullfont
6830             \hrule \@height \l_tmpa_dim
6831                 \@depth \c_zero_dim
6832                 \@width \c_zero_dim
6833         }
6834         \right .
6835         \c_math_toggle_token
6836     }
6837     { #2 }
6838     { }
6839 }

```

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`). The argument `#3` is the subscript and `#4` is the superscript.

```

6840 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
6841 {
6842     \pgfnode
6843     { rectangle }
6844     { west }
6845     {

```

```

6846 \nullfont
6847 \c_math_toggle_token
6848 \tl_if_empty:NF \l_@@_delimiters_color_tl
6849 { \color { \l_@@_delimiters_color_tl } }
6850 \left .
6851 \vcenter
6852 {
6853 \nullfont
6854 \hrule \@height \l_tmpa_dim
6855 \depth \c_zero_dim
6856 \width \c_zero_dim
6857 }
6858 \right #1
6859 \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
6860 ^ { \smash { #4 } }
6861 \c_math_toggle_token
6862 }
6863 { #2 }
6864 { }
6865 }

```

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.

```

6866 \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.

```

6867 \bool_new:N \c_@@_footnote_bool
6868 \@@_msg_new:nnn { Unknown~key~for~package }
6869 {
6870 The~key~'\l_keys_key_str'~is~unknown. \\\
6871 If~you~go~on,~it~will~be~ignored. \\\
6872 For~a~list~of~the~available~keys,~type~H~<return>.
6873 }
6874 {
6875 The~available~keys~are~(in~alphabetic~order):~
6876 footnote,~
6877 footnotehyper,~
6878 renew-dots,~and
6879 renew-matrix.
6880 }

```

Maybe we will completely delete the key 'transparent' in a future version.

```

6881 \@@_msg_new:nn { Key~transparent }
6882 {
6883 The~key~'transparent'~is~now~obsolete~(because~it's~name~
6884 is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
6885 and~'renew-matrix'.~However,~you~can~go~on.
6886 }
6887 \keys_define:nn { NiceMatrix / Package }
6888 {
6889 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
6890 renew-dots .value_forbidden:n = true ,

```

```

6891   renew-matrix .code:n = \@@_renew_matrix: ,
6892   renew-matrix .value_forbidden:n = true ,
6893   transparent .code:n =
6894   {
6895     \@@_renew_matrix:
6896     \bool_set_true:N \l_@@_renew_dots_bool
6897     \@@_error:n { Key-transparent }
6898   } ,
6899   transparent .value_forbidden:n = true,
6900   footnote .bool_set:N = \c_@@_footnote_bool ,
6901   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
6902   unknown .code:n = \@@_error:n { Unknown-key-for-package }
6903 }
6904 \ProcessKeysOptions { NiceMatrix / Package }

6905 \@@_msg_new:nn { footnote-with-footnotehyper-package }
6906 {
6907   You~can't~use~the~option~'footnote'~because~the~package~
6908   footnotehyper~has~already~been~loaded.~
6909   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
6910   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6911   of~the~package~footnotehyper.\\
6912   If~you~go~on,~the~package~footnote~won't~be~loaded.
6913 }

6914 \@@_msg_new:nn { footnotehyper-with-footnote-package }
6915 {
6916   You~can't~use~the~option~'footnotehyper'~because~the~package~
6917   footnote~has~already~been~loaded.~
6918   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
6919   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6920   of~the~package~footnote.\\
6921   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
6922 }

6923 \bool_if:NT \c_@@_footnote_bool
6924 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6925   \@ifclassloaded { beamer }
6926   { \bool_set_false:N \c_@@_footnote_bool }
6927   {
6928     \@ifpackageloaded { footnotehyper }
6929     { \@@_error:n { footnote-with-footnotehyper-package } }
6930     { \usepackage { footnote } }
6931   }
6932 }

6933 \bool_if:NT \c_@@_footnotehyper_bool
6934 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6935   \@ifclassloaded { beamer }
6936   { \bool_set_false:N \c_@@_footnote_bool }
6937   {
6938     \@ifpackageloaded { footnote }
6939     { \@@_error:n { footnotehyper-with-footnote-package } }
6940     { \usepackage { footnotehyper } }
6941   }
6942   \bool_set_true:N \c_@@_footnote_bool
6943 }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

Error messages of the package

The following message will be deleted when we will delete the key `except-corners` for the command `\arraycolor`.

```

6944 \@@_msg_new:nn { key~except-corners }
6945 {
6946   The~key~'except-corners'~has~been~deleted~for~the~command~\token_to_str:N
6947   \arraycolor\ in~the~\token_to_str:N \CodeBefore.~You~should~instead~use~
6948   the~key~'corners'~in~your~\@@_full_name_env:.\\
6949   If~you~go~on,~this~key~will~be~ignored.
6950 }
6951 \seq_new:N \c_@@_types_of_matrix_seq
6952 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
6953 {
6954   NiceMatrix ,
6955   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
6956 }
6957 \seq_set_map_x:Nn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
6958 { \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`.

```

6959 \cs_new_protected:Npn \@@_error_too_much_cols:
6960 {
6961   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
6962   {
6963     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
6964     { \@@_fatal:n { too~much~cols~for~matrix } }
6965     {
6966       \bool_if:NF \l_@@_last_col_without_value_bool
6967       { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
6968     }
6969   }
6970   { \@@_fatal:n { too~much~cols~for~array } }
6971 }

```

The following command must *not* be protected since it's used in an error message.

```

6972 \cs_new:Npn \@@_message_hdotsfor:
6973 {
6974   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
6975   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
6976 }
6977 \@@_msg_new:nn { negative~weight }
6978 {
6979   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
6980   the~value~'#1'.~If~you~go~on,~the~absolute~value~will~be~used.
6981 }
6982 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
6983 {
6984   You~try~to~use~more~columns~than~allowed~by~your~
6985   \@@_full_name_env:. \@@_message_hdotsfor:\ The~maximal~number~of~
6986   columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
6987   exterior~columns).~This~error~is~fatal.
6988 }

```

```

6989 \@@_msg_new:nn { too-much-cols-for-matrix }
6990 {
6991   You-try-to-use-more-columns-than-allowed-by-your~
6992   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
6993   number-of-columns-for-a-matrix-is-fixed-by-the-LaTeX-counter~
6994   'MaxMatrixCols'.~Its-actual-value-is~\int_use:N \c@MaxMatrixCols.~
6995   This-error-is-fatal.
6996 }

```

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`.

```

6997 \@@_msg_new:nn { too-much-cols-for-array }
6998 {
6999   You-try-to-use-more-columns-than-allowed-by-your~
7000   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
7001   \int_use:N \g_@@_static_num_of_col_int\
7002   ~(plus-the-potential-exterior-ones).~
7003   This-error-is-fatal.
7004 }

7005 \@@_msg_new:nn { last-col-not-used }
7006 {
7007   The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
7008   in~your~\@@_full_name_env:~.~However,~you~can~go~on.
7009 }

7010 \@@_msg_new:nn { columns-not-used }
7011 {
7012   The-preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7013   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
7014   You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
7015 }

7016 \@@_msg_new:nn { in-first-col }
7017 {
7018   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
7019   If~you~go~on,~this~command~will~be~ignored.
7020 }

7021 \@@_msg_new:nn { in-last-col }
7022 {
7023   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
7024   If~you~go~on,~this~command~will~be~ignored.
7025 }

7026 \@@_msg_new:nn { in-first-row }
7027 {
7028   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
7029   If~you~go~on,~this~command~will~be~ignored.
7030 }

7031 \@@_msg_new:nn { in-last-row }
7032 {
7033   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
7034   If~you~go~on,~this~command~will~be~ignored.
7035 }

7036 \@@_msg_new:nn { double-closing-delimiter }
7037 {
7038   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
7039   delimiter.~This~delimiter~will~be~ignored.
7040 }

7041 \@@_msg_new:nn { delimiter~after~opening }
7042 {
7043   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
7044   delimiter.~This~delimiter~will~be~ignored.
7045 }

```

```

7046 \@@_msg_new:nn { bad-option-for-line-style }
7047 {
7048     Since-you-haven't-loaded-Tikz,~the-only~value~you~can~give~to~'line-style'~
7049     is~'standard'.~If-you-go-on,~this-key~will~be~ignored.
7050 }
7051 \@@_msg_new:nn { Unknown-key-for-xdots }
7052 {
7053     As-for~now,~there~is~only~three~keys~available~here:~'color',~'line-style'~
7054     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
7055     this-key~will~be~ignored.
7056 }
7057 \@@_msg_new:nn { Unknown-key-for-rowcolors }
7058 {
7059     As-for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
7060     (and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
7061     this-key~will~be~ignored.
7062 }
7063 \@@_msg_new:nn { ampersand-in-light-syntax }
7064 {
7065     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
7066     ~you~have~used~the~key~'light-syntax'.~This~error~is~fatal.
7067 }
7068 \@@_msg_new:nn { SubMatrix-too-large }
7069 {
7070     Your~command~\token_to_str:N \SubMatrix\
7071     can't~be~drawn~because~your~matrix~is~too~small.\\
7072     If~you~go~on,~this~command~will~be~ignored.
7073 }
7074 \@@_msg_new:nn { double-backslash-in-light-syntax }
7075 {
7076     You~can't~use~\token_to_str:N \\~to~separate~rows~because~you~have~used~
7077     the~key~'light-syntax'.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
7078     (set~by~the~key~'end-of-row').~This~error~is~fatal.
7079 }
7080 \@@_msg_new:nn { standard-cline-in-document }
7081 {
7082     The~key~'standard-cline'~is~available~only~in~the~preamble.\\
7083     If~you~go~on~this~command~will~be~ignored.
7084 }
7085 \@@_msg_new:nn { bad-value-for-baseline }
7086 {
7087     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
7088     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
7089     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
7090     If~you~go~on,~a~value~of~1~will~be~used.
7091 }
7092 \@@_msg_new:nn { Invalid-name-format }
7093 {
7094     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
7095     \SubMatrix.\\
7096     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
7097     If~you~go~on,~this~key~will~be~ignored.
7098 }
7099 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
7100 {
7101     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
7102     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
7103     number~is~not~valid.~If~you~go~on,~it~will~be~ignored.
7104 }

```



```

7105 \@@_msg_new:nn { impossible-delimiter }
7106 {
7107   It's-impossible-to-draw-the-#1-delimiter-of-your-
7108   \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty-
7109   in-that-column.
7110   \bool_if:NT \l_@@_submatrix_slim_bool
7111   { ~Maybe-you-should-try-without-the-key-'slim'. } \
7112   If-you-go-on,~this-\token_to_str:N \SubMatrix\ will-be-ignored.
7113 }
7114 \@@_msg_new:nn { width-without-X-columns }
7115 {
7116   You-have-used-the-key-'width'~but-you-have-put-no-'X'-column. \
7117   If-you-go-on,~that-key-will-be-ignored.
7118 }
7119 \@@_msg_new:nn { empty-environment }
7120 { Your-\@@_full_name_env:\ is-empty.~This-error-is-fatal. }
7121 \@@_msg_new:nn { Delimiter-with-small }
7122 {
7123   You-can't-put-a-delimiter-in-the-preamble-of-your-\@@_full_name_env:\
7124   because-the-key-'small'~is-in-force.\
7125   This-error-is-fatal.
7126 }
7127 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
7128 {
7129   Your-command-\token_to_str:N\line\{#1\}\{#2\}~in-the-'code-after'-
7130   can't-be-executed-because-a-cell-doesn't-exist.\
7131   If-you-go-on~this-command-will-be-ignored.
7132 }
7133 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
7134 {
7135   The-name-'#1'~is-already-used-for-a-\token_to_str:N \SubMatrix\
7136   in-this-\@@_full_name_env:.\
7137   If-you-go-on,~this-key-will-be-ignored.\
7138   For-a-list-of-the-names-already-used,~type-H~<return>.
7139 }
7140 {
7141   The-names-already-defined-in-this-\@@_full_name_env:\ are:~
7142   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
7143 }
7144 \@@_msg_new:nn { r-or-l-with-preamble }
7145 {
7146   You-can't-use-the-key-'l_keys_key_str'~in-your-\@@_full_name_env:~
7147   You-must-specify-the-alignment-of-your-columns-with-the-preamble-of-
7148   your-\@@_full_name_env:.\
7149   If-you-go-on,~this-key-will-be-ignored.
7150 }
7151 \@@_msg_new:nn { Hdotsfor~in~col-0 }
7152 {
7153   You-can't-use-\token_to_str:N \Hdotsfor\ in-an-exterior-column-of-
7154   the-array.~This-error-is-fatal.
7155 }
7156 \@@_msg_new:nn { bad-corner }
7157 {
7158   #1-is-an-incorrect-specification-for-a-corner~(in-the-keys-
7159   'corners'~and~'except-corners').~The-available~
7160   values-are:~NW,~SW,~NE~and~SE.\
7161   If-you-go-on,~this-specification-of-corner-will-be-ignored.
7162 }
7163 \@@_msg_new:nn { bad-border }
7164 {

```

```

7165     #1~is~an~incorrect~specification~for~a~border~(in~the~key~
7166     'borders'~of~the~command~\token_to_str:N \Block).~The~available~
7167     values~are:~left,~right,~top~and~bottom.\\
7168     If~you~go~on,~this~specification~of~border~will~be~ignored.
7169 }

7170 \@@_msg_new:nn { tikz~key~without~tikz }
7171 {
7172     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
7173     \Block'~because~you~have~not~loaded~Tikz.~
7174     If~you~go~on,~this~key~will~be~ignored.
7175 }

7176 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
7177 {
7178     In~the~\@@_full_name_env:,~you~must~use~the~key~
7179     'last~col'~without~value.\\
7180     However,~you~can~go~on~for~this~time~
7181     (the~value~'\l_keys_value_tl'~will~be~ignored).
7182 }

7183 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
7184 {
7185     In~\NiceMatrixoptions,~you~must~use~the~key~
7186     'last~col'~without~value.\\
7187     However,~you~can~go~on~for~this~time~
7188     (the~value~'\l_keys_value_tl'~will~be~ignored).
7189 }

7190 \@@_msg_new:nn { Block~too~large~1 }
7191 {
7192     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
7193     too~small~for~that~block. \\
7194 }

7195 \@@_msg_new:nn { Block~too~large~2 }
7196 {
7197     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7198     \g_@@_static_num_of_col_int\
7199     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
7200     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
7201     (&)~at~the~end~of~the~first~row~of~your~
7202     \@@_full_name_env:.\
7203     If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
7204 }

7205 \@@_msg_new:nn { unknown~column~type }
7206 {
7207     The~column~type~'#1'~in~your~\@@_full_name_env:\
7208     is~unknown. \\
7209     This~error~is~fatal.
7210 }

7211 \@@_msg_new:nn { tabularnote~forbidden }
7212 {
7213     You~can't~use~the~command~\token_to_str:N\ tabularnote\
7214     ~in~a~\@@_full_name_env:~.~This~command~is~available~only~in~
7215     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
7216     If~you~go~on,~this~command~will~be~ignored.
7217 }

7218 \@@_msg_new:nn { borders~forbidden }
7219 {
7220     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
7221     because~the~option~'rounded~corners'~
7222     is~in~force~with~a~non~zero~value.\\
7223     If~you~go~on,~this~key~will~be~ignored.
7224 }

```

```

7225 \@@_msg_new:nn { bottomrule-without-booktabs }
7226 {
7227     You~can't~use~the~key~'\tabular/bottomrule'~because~you~haven't~
7228     loaded~'\booktabs'.\\
7229     If~you~go~on,~this~key~will~be~ignored.
7230 }
7231 \@@_msg_new:nn { enumitem~not~loaded }
7232 {
7233     You~can't~use~the~command~\token_to_str:N\tabularnote\
7234     ~because~you~haven't~loaded~'\enumitem'.\\
7235     If~you~go~on,~this~command~will~be~ignored.
7236 }
7237 \@@_msg_new:nn { Wrong~last~row }
7238 {
7239     You~have~used~'\last-row=\int_use:N \l_@@_last_row_int'~but~your~
7240     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
7241     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
7242     last~row.~You~can~avoid~this~problem~by~using~'\last-row'~
7243     without~value~(more~compilations~might~be~necessary).
7244 }
7245 \@@_msg_new:nn { Yet~in~env }
7246 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }
7247 \@@_msg_new:nn { Outside~math~mode }
7248 {
7249     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
7250     (and~not~in~\token_to_str:N \vcenter).\\
7251     This~error~is~fatal.
7252 }
7253 \@@_msg_new:nn { One~letter~allowed }
7254 {
7255     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
7256     If~you~go~on,~it~will~be~ignored.
7257 }
7258 \@@_msg_new:nn { varwidth~not~loaded }
7259 {
7260     You~can't~use~the~column~type~'V'~because~'\varwidth'~is~not~
7261     loaded.\\
7262     If~you~go~on,~your~column~will~behave~like~'p'.
7263 }
7264 \@@_msg_new:nnn { Unknown~key~for~Block }
7265 {
7266     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
7267     \Block.\\ If~you~go~on,~it~will~be~ignored. \\
7268     For~a~list~of~the~available~keys,~type~H~<return>.
7269 }
7270 {
7271     The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
7272     hvlines,~l,~line~width,~name,~rounded~corners,~r,~t~and~tikz.
7273 }
7274 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
7275 {
7276     The~key~'\l_keys_key_str'~is~unknown.\\
7277     If~you~go~on,~it~will~be~ignored. \\
7278     For~a~list~of~the~available~keys~in~\token_to_str:N
7279     \CodeAfter,~type~H~<return>.
7280 }
7281 {
7282     The~available~keys~are~(in~alphabetic~order):~
7283     delimiters/color,~
7284     rules~(with~the~subkeys~'\color'~and~'\width'),~

```

```

7285     sub-matrix~(several~subkeys)~
7286     and~xdots~(several~subkeys).~
7287     The~latter~is~for~the~command~\token_to_str:N \line.
7288 }

7289 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
7290 {
7291     The~key~'\l_keys_key_str'~is~unknown.\\
7292     If~you~go~on,~this~key~will~be~ignored. \\
7293     For~a~list~of~the~available~keys~in~\token_to_str:N
7294     \SubMatrix,~type~H~<return>.
7295 }
7296 {
7297     The~available~keys~are~(in~alphabetic~order):~
7298     'delimiters/color',~
7299     'extra-height',~
7300     'hlines',~
7301     'hvlines',~
7302     'left-xshift',~
7303     'name',~
7304     'right-xshift',~
7305     'rules'~(with~the~subkeys~'color'~and~'width'),~
7306     'slim',~
7307     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
7308     and~'right-xshift').\\
7309 }

7310 \@@_msg_new:nnn { Unknown~key~for~notes }
7311 {
7312     The~key~'\l_keys_key_str'~is~unknown.\\
7313     If~you~go~on,~it~will~be~ignored. \\
7314     For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
7315 }
7316 {
7317     The~available~keys~are~(in~alphabetic~order):~
7318     bottomrule,~
7319     code-after,~
7320     code-before,~
7321     enumitem-keys,~
7322     enumitem-keys-para,~
7323     para,~
7324     label-in-list,~
7325     label-in-tabular~and~
7326     style.
7327 }

7328 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
7329 {
7330     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7331     \token_to_str:N \RowStyle. \\
7332     If~you~go~on,~it~will~be~ignored. \\
7333     For~a~list~of~the~available~keys,~type~H~<return>.
7334 }
7335 {
7336     The~available~keys~are~(in~alphabetic~order):~
7337     'bold',~
7338     'cell-space-top-limit',~
7339     'cell-space-bottom-limit',~
7340     'cell-space-limits',~
7341     'color',~
7342     'nb-rows'~and~
7343     'rowcolor'.
7344 }

7345 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
7346 {

```

```

7347 The-key~'\l_keys_key_str'~is-unknown~for~the-command~
7348 \token_to_str:N \NiceMatrixOptions. \\
7349 If~you-go-on,~it~will~be~ignored. \\
7350 For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7351 }
7352 {
7353 The~available~keys~are~(in~alphabetic~order):~
7354 allow-duplicate-names,~
7355 cell-space-bottom-limit,~
7356 cell-space-limits,~
7357 cell-space-top-limit,~
7358 code-for-first-col,~
7359 code-for-first-row,~
7360 code-for-last-col,~
7361 code-for-last-row,~
7362 corners,~
7363 create-extra-nodes,~
7364 create-medium-nodes,~
7365 create-large-nodes,~
7366 delimiters~(several~subkeys),~
7367 end-of-row,~
7368 first-col,~
7369 first-row,~
7370 hlines,~
7371 hvlines,~
7372 last-col,~
7373 last-row,~
7374 left-margin,~
7375 letter-for-dotted-lines,~
7376 light-syntax,~
7377 notes~(several~subkeys),~
7378 nullify-dots,~
7379 renew-dots,~
7380 renew-matrix,~
7381 right-margin,~
7382 rules~(with~the~subkeys~'color'~and~'width'),~
7383 small,~
7384 sub-matrix~(several~subkeys),
7385 vlines,~
7386 xdots~(several~subkeys).
7387 }
7388 @@_msg_new:nnn { Unknown~key~for~NiceArray }
7389 {
7390 The-key~'\l_keys_key_str'~is-unknown~for~the~environment~
7391 \{NiceArray\}. \\
7392 If~you-go-on,~it~will~be~ignored. \\
7393 For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7394 }
7395 {
7396 The~available~keys~are~(in~alphabetic~order):~
7397 b,~
7398 baseline,~
7399 c,~
7400 cell-space-bottom-limit,~
7401 cell-space-limits,~
7402 cell-space-top-limit,~
7403 code-after,~
7404 code-for-first-col,~
7405 code-for-first-row,~
7406 code-for-last-col,~
7407 code-for-last-row,~
7408 colortbl-like,~
7409 columns-width,~

```

```

7410     corners,~
7411     create-extra-nodes,~
7412     create-medium-nodes,~
7413     create-large-nodes,~
7414     delimiters/color,~
7415     extra-left-margin,~
7416     extra-right-margin,~
7417     first-col,~
7418     first-row,~
7419     hlines,~
7420     hvlines,~
7421     last-col,~
7422     last-row,~
7423     left-margin,~
7424     light-syntax,~
7425     name,~
7426     notes/bottomrule,~
7427     notes/para,~
7428     nullify-dots,~
7429     renew-dots,~
7430     right-margin,~
7431     rules~(with~the~subkeys~'color'~and~'width'),~
7432     small,~
7433     t,~
7434     tabularnote,~
7435     vlines,~
7436     xdots/color,~
7437     xdots/shorten~and~
7438     xdots/line-style.
7439 }

```

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).

```

7440 \@@_msg_new:nnn { Unknown-key-for-NiceMatrix }
7441 {
7442   The~key~'\l_keys_key_str'~is~unknown~for~the~
7443   \@@_full_name_env:. \\\
7444   If~you~go~on,~it~will~be~ignored. \\\
7445   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7446 }
7447 {
7448   The~available~keys~are~(in~alphabetic~order):~
7449   b,~
7450   baseline,~
7451   c,~
7452   cell-space-bottom-limit,~
7453   cell-space-limits,~
7454   cell-space-top-limit,~
7455   code-after,~
7456   code-for-first-col,~
7457   code-for-first-row,~
7458   code-for-last-col,~
7459   code-for-last-row,~
7460   colortbl-like,~
7461   columns-width,~
7462   corners,~
7463   create-extra-nodes,~
7464   create-medium-nodes,~
7465   create-large-nodes,~
7466   delimiters~(several~subkeys),~
7467   extra-left-margin,~
7468   extra-right-margin,~
7469   first-col,~

```

```

7470 first-row,~
7471 hlines,~
7472 hvlines,~
7473 l,~
7474 last-col,~
7475 last-row,~
7476 left-margin,~
7477 light-syntax,~
7478 name,~
7479 nullify-dots,~
7480 r,~
7481 renew-dots,~
7482 right-margin,~
7483 rules~(with~the~subkeys~'color'~and~'width'),~
7484 small,~
7485 t,~
7486 vlines,~
7487 xdots/color,~
7488 xdots/shorten~and~
7489 xdots/line-style.
7490 }

7491 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
7492 {
7493   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7494   \{NiceTabular\}. \\
7495   If~you~go~on,~it~will~be~ignored. \\
7496   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7497 }
7498 {
7499   The~available~keys~are~(in~alphabetic~order):~
7500   b,~
7501   baseline,~
7502   c,~
7503   cell-space-bottom-limit,~
7504   cell-space-limits,~
7505   cell-space-top-limit,~
7506   code-after,~
7507   code-for-first-col,~
7508   code-for-first-row,~
7509   code-for-last-col,~
7510   code-for-last-row,~
7511   colortbl-like,~
7512   columns-width,~
7513   corners,~
7514   create-extra-nodes,~
7515   create-medium-nodes,~
7516   create-large-nodes,~
7517   extra-left-margin,~
7518   extra-right-margin,~
7519   first-col,~
7520   first-row,~
7521   hlines,~
7522   hvlines,~
7523   last-col,~
7524   last-row,~
7525   left-margin,~
7526   light-syntax,~
7527   name,~
7528   notes/bottomrule,~
7529   notes/para,~
7530   nullify-dots,~
7531   renew-dots,~
7532   right-margin,~

```

```

7533     rules~(with~the~subkeys~'color'~and~'width'),~
7534     t,~
7535     tabularnote,~
7536     vl原因,~
7537     xdots/color,~
7538     xdots/shorten~and~
7539     xdots/line~style.
7540 }

7541 \@@_msg_new:nnn { Duplicate~name }
7542 {
7543     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
7544     the~same~environment~name~twice.~You~can~go~on,~but,~
7545     maybe,~you~will~have~incorrect~results~especially~
7546     if~you~use~'columns~width=auto'.~If~you~don't~want~to~see~this~
7547     message~again,~use~the~key~'allow~duplicate~names'~in~
7548     '\token_to_str:N \NiceMatrixOptions'.\\
7549     For~a~list~of~the~names~already~used,~type~H~<return>. \\
7550 }
7551 {
7552     The~names~already~defined~in~this~document~are:~
7553     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
7554 }

7555 \@@_msg_new:nn { Option~auto~for~columns~width }
7556 {
7557     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
7558     If~you~go~on,~the~key~will~be~ignored.
7559 }

```

19 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 StackExchange⁷³, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁷⁴

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

⁷³cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁷⁴Before this version, there was an error when using `nicematrix` with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by `nicematrix` because they use the options `overlay` and `remember picture`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`).

Possibility to draw vertical dotted lines to separate columns with the specifier “:” in the preamble (similar to the classical specifier “|” and the specifier “:” of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “:” in the preamble.

Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.

Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of “|”) as `\hdotsfor` does.

Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.

Error message when the user gives an incorrect value for `last-row`.

A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol “:” (in the preamble of the array) and `\line` in `code-after`).

The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.

The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by `colortbl`.

Correction of a bug: it was not possible to use the colon “:” in the preamble of an array when `pdf \LaTeX` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

The option `columns-width=auto` doesn’t need any more a second compilation.

The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).

The previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange⁷⁵, optimization of Tikz externalization is disabled in the environments of `nicematrix` when the class `standalone` or the package `standalone` is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

Changes between version 3.5 and 3.6

LaTeX counters `iRow` and `jCol` available in the cells of the array.

Addition of `\normalbaselines` before the construction of the array: in environments like `{align}` of `amsmath` the value of `\baselineskip` is changed and if the options `first-row` and `last-row` were used in an environment of `nicematrix`, the position of the delimiters was wrong.

A warning is written in the `.log` file if an obsolete environment is used.

There is no longer artificial errors `Duplicate~name` in the environments of `amsmath`.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.

New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programming for the command `\Block` when the block has only one row. With this programming, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with `\multicolumn`. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands `\NiceMatrixLastEnv` and `\OnlyMainNiceMatrix`.

New options `create-medium-nodes` and `create-large-nodes`.

Changes between version 3.9 and 3.10

New option `light-syntax` (and `end-of-row`).

New option `dotted-lines-margin` for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

Correction of a bug linked to `first-row` and `last-row`.

⁷⁵cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -block and, if the creation of the “medium nodes” is required, a node $i-j$ -block-medium is created.

If the user tries to use more columns than allowed by its environment, an error is raised by `nicematrix` (instead of a low-level error).

The package must be loaded with the option `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the “last row”.

The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customisation of the dotted lines.

In the environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, etc.), it’s possible to use the options `l` (=L) or `r` (=R) to specify the type of the columns.

The starred versions of the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots` and `\Iddots` are deprecated since the version 3.1 of `nicematrix`. Now, one should load `nicematrix` with the option `starred-commands` to avoid an error at the compilation.

The code of `nicematrix` no longer uses Tikz but only PGF. By default, Tikz is *not* loaded by `nicematrix`.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on [stackoverflow](#)).

Better error messages when the user uses `&` or `\\` when `light-syntax` is in force.

Changes between versions 3.14 and 3.15

It’s possible to put labels on the dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and the command `\line` in the `code-after` with the tokens `_` and `^`.

The option `baseline` is now available in all the environments of `nicematrix`. Before, it was available only in `{NiceArray}`.

New keyword `\CodeAfter` (in the environments of `nicematrix`).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`

Commands to color cells, rows and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`

New command `\diagbox`

The key `hvline` don’t draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It’s now possible to write `\begin{pNiceMatrix}a&b\\c&d\end{pNiceMatrix}`² with the expected result.

Changes between versions 4.2 and 4.3

The horizontal centering of the content of a `\Block` is correct even when an instruction such as `!\qqquad` is used in the preamble of the array.

It's now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.

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

Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).

Environment `{NiceTabular*}`

Command `\Vdotsfor` similar to `\Hdotsfor`

The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.

Key `respect-blocks` for `\rowcolors` (with a *s*) in the `code-before`.

The variable `\g_nicematrix_code_before_tl` is now public.

The key `baseline` may take in as value an expression of the form *line-i* to align the `\hline` in the row *i*.

The key `hvlines-except-corners` may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys `c`, `r` and `l` for the command `\Block`.

It's possible to use the key `draw-first` with `\Ddots` and `\Iddots` to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key `tabularnote`.

Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put `\omit` before `\CodeAfter`.

Correction of a bug: the tabular notes `\tabularnotes` were not composed when present in a block (except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.

New command `\NotEmpty`.

Changes between versions 5.6 and 5.7

New key `delimiters-color`

Keys `fill`, `draw` and `line-width` for the command `\Block`.

Changes between versions 5.7 and 5.8

Keys `cols` and `restart` of the command `\rowcolors` in the `code-before`.

Modification of the behaviour of `\\` in the columns of type `p`, `m` or `b` (for a behaviour similar to the environments of `array`).

Better error messages for the command `\Block`.

Changes between versions 5.8 and 5.9

Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.

New key `cell-space-limits`.

Changes between versions 5.9 and 5.10

New command `\SubMatrix` available in the `\CodeAfter`.

It's possible to provide options (between brackets) to the keyword `\CodeAfter`.

A (non fatal) error is raised when the key `transparent`, which is deprecated, is used.

Changes between versions 5.10 and 5.11

It's now possible, in the `code-before` and in the `\CodeAfter`, to use the syntax `|(i-|j)` for the Tikz node at the intersection of the (potential) horizontal rule number i and the (potential) vertical rule number j .

Changes between versions 5.11 and 5.12

Keywords `\CodeBefore` and `\Body` (alternative syntax to the key `code-before`).

New key `delimiters/max-width`.

New keys `hlines`, `vlines` and `hvlines` for the command `\SubMatrix` in the `\CodeAfter`.

New key `rounded-corners` for the command `\Block`.

Changes between versions 5.12 and 5.13

New command `\arraycolor` in the `\CodeBefore` (with its key `except-corners`).

New key `borders` for the command `\Block`.

New command `\Hline` (for horizontal rules not drawn in the blocks).

The keys `vlines` and `hlines` takes in as value a (comma-separated) list of numbers (for the rules to draw).

Changes between versions 5.13 and 5.14

Nodes of the form (1.5) , (2.5) , (3.5) , etc.

Keys `t` and `b` for the command `\Block`.

Key `corners`.

Changes between versions 5.14 and 5.15

Key `hvlines` for the command `\Block`.

The commands provided by `nicematrix` to color cells, rows and columns don't color the cells which are in the “corners” (when the key `corner` is used).

It's now possible to specify delimiters for submatrices in the preamble of an environment.

The version 5.15b is compatible with the version 3.0+ of `siunitx` (previous versions were not).

Changes between versions 5.15 and 5.16

It's now possible to use the cells corresponding to the contents of the nodes (of the form `i-j`) in the `\CodeBefore` when the key `create-cell-nodes` of that `\CodeBefore` is used. The medium and the large nodes are also available if the corresponding keys are used.

Changes between versions 5.16 and 5.17

The key `define-L-C-R` (only available at load-time) now raises a (non fatal) error.

Keys `L`, `C` and `R` for the command `\Block`.

Key `hvlines-except-borders`.

It's now possible to use a key `l`, `r` or `c` with the command `\pAutoNiceMatrix` (and the similar ones).

Changes between versions 5.17 and 5.18

New command `\RowStyle`

Changes between versions 5.18 and 5.19

New key `tikz` for the command `\Block`.

Changes between versions 5.19 and 6.0

Columns `X` and environment `{NiceTabularX}`.

Command `\rowlistcolors` available in the `\CodeBefore`.

In columns with fixed width, the blocks are composed as paragraphs (wrapping of the lines).

The key `define-L-C-R` has been deleted.

Changes between versions 6.0 and 6.1

Better computation of the widths of the `X` columns.

Key `\color` for the command `\RowStyle`.

Changes between versions 6.1 and 6.2

Better compatibility with the classes `revtex4-1` and `revtex4-2`.

Key `vlines-in-sub-matrix`.

Changes between versions 6.2 and 6.3

Keys `nb-rows`, `rowcolor` and `bold` for the command `\RowStyle`

Key `name` for the command `\Block`.

Support for the columns `V` of `varwidth`.

Index

The *italic* numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
@@ commands:	
\@@_Block:	1252, 5548
\@@_Block_i	5553, 5554, 5558
\@@_Block_ii:nnnnn	5558, 5559
\@@_Block_iv:nnnnn	5596, 5600
\@@_Block_iv:nnnnnn	5815, 5817
\@@_Block_v:nnnnn	5597, 5719
\@@_Block_v:nnnnnn	5844, 5847
\@@_Cdots	1166, 1243, 4003
\g_@@_Cdots_lines_tl	1273, 3197
\@@_CodeAfter:	1256, 6422
\@@_CodeAfter_i:	900, 2845, 2890, 6423
\@@_CodeAfter_ii:n	6422, 6423, 6424, 6434
\@@_CodeAfter_iv:n	6427, 6429
\@@_CodeAfter_keys:	3137, 3157
\@@_CodeBefore:w	1415, 1417
\@@_CodeBefore_keys:	1395, 1412
\@@_Ddots	1168, 1245, 4035
\g_@@_Ddots_lines_tl	1276, 3195
\g_@@_HVdotsfor_lines_tl	1278, 3193, 4119, 4196, 6974
\@@_Hdotsfor:	1171, 1250, 4095
\@@_Hdotsfor:nnnn	4121, 4134
\@@_Hdotsfor_i	4104, 4110, 4117
\@@_Hline:	1248, 5097
\@@_Hline_i:n	5097, 5098, 5104
\@@_Hline_ii:nn	5101, 5104
\@@_Hline_iii:n	5102, 5105
\@@_Hspace:	1249, 4089
\@@_Iddots	1169, 1246, 4059
\g_@@_Iddots_lines_tl	1277, 3196
\@@_Ldots	1165, 1170, 1242, 3987
\g_@@_Ldots_lines_tl	1274, 3198
\l_@@_Matrix_bool	263, 1691, 1715, 2451, 2789, 2793, 2801, 2958
\l_@@_NiceArray_bool	260, 419, 1319, 1559, 1631, 1754, 1761, 1773, 2451, 2777, 2789, 2793, 2801, 2934, 4914, 4918, 5062, 5072, 5083, 5087, 6326
\g_@@_NiceMatrixBlock_int	245, 5308, 5313, 5316, 5327
\l_@@_NiceTabular_bool	184, 261, 905, 1093, 1394, 1397, 1526, 1664, 1668, 1762, 1774, 2778, 2989, 3010, 3019, 3136, 3139, 5629, 5728, 6334
\@@_NotEmpty:	1258, 2981
\@@_OnlyMainNiceMatrix:n	1254, 4752
\@@_OnlyMainNiceMatrix_i:n	4755, 4762, 4765
\@@_RowStyle:n	1259, 4347
\@@_S:	209, 1804
\@@_SubMatrix	3118, 6573
\@@_SubMatrix_in_code_before	1387, 6553
\@@_V:	1719, 1800
\@@_Vdots	1167, 1244, 4019
\g_@@_Vdots_lines_tl	1275, 3194
\@@_Vdotsfor:	1251, 4194
\@@_Vdotsfor:nnnn	4198, 4209
\@@_W:	1718, 1803, 2308
\@@_X	1812, 3005
\l_@@_X_column_bool	265, 2208, 5594
\l_@@_X_columns_aux_bool	292, 1584, 2197
\l_@@_X_columns_dim	293, 1585, 1594, 1600, 2200
\@@_actually_color:	1396, 4434
\@@_actually_diagbox:nnnnnn	5607, 5916, 6346, 6363
\@@_actually_draw_Cdots:	3563, 3567
\@@_actually_draw_Ddots:	3713, 3717
\@@_actually_draw_Iddots:	3761, 3765
\@@_actually_draw_Ldots:	3524, 3528, 4185
\@@_actually_draw_Vdots:	3645, 3649, 4260
\@@_adapt_S_column:	217, 222, 235, 1525
\@@_add_to_colors_seq:nn	4420, 4432, 4433, 4457, 4466, 4475, 4484, 4591
\@@_adjust_pos_of_blocks_seq:	3115, 3159
\@@_adjust_pos_of_blocks_seq_i:nnnnn	3162, 3164
\@@_adjust_size_box:	978, 1005, 2043, 2368, 2869, 2914
\@@_adjust_to_submatrix:nn	3408, 3511, 3550, 3631, 3706, 3754
\@@_adjust_to_submatrix:nnnnnn	3415, 3417
\@@_after_array:	1700, 3023
\g_@@_after_col_zero_bool	294, 1133, 2846, 4101
\@@_analyze_end:Nn	2599, 2644
\l_@@_argspec_tl	3985, 3986, 3987, 4003, 4019, 4035, 4059, 4115, 4116, 4117, 4192, 4193, 4194, 4270, 4271, 4272, 6571, 6572, 6573
\@@_array:	1091, 2600, 2627
\@@_arraycolor	1384, 4521
\c_@@_arydshln_loaded_bool	24, 36, 1830
\l_@@_auto_columns_width_bool	522, 668, 2711, 2715, 5303
\g_@@_aux_found_bool	1290, 1295, 1419, 1547, 1550
\g_@@_aux_tl	266, 1553, 1582, 1707, 3032, 3046, 3054, 3146, 5180
\l_@@_bar_at_end_of_pream_bool	1755, 1890, 2781
\l_@@_baseline_tl	513, 514, 661, 662, 663, 664, 1100, 1633, 2404, 2416, 2421, 2423, 2428, 2433, 2523, 2524, 2528, 2533, 2535, 2540
\@@_begin_of_NiceMatrix:nn	2956, 2977
\@@_begin_of_row:	903, 928, 1202, 2847
\l_@@_block_auto_columns_width_bool	1539, 2716, 5296, 5301, 5311, 5321
\g_@@_block_box_int	338, 1519, 5602, 5616, 5618, 5672, 5684, 5696, 5705, 5715
\l_@@_block_name_str	5807, 5863, 5940, 5943, 5948
\@@_block_tikz:nnnnn	5904, 6236
\g_@@_blocks_dp_dim	256, 986, 989, 990, 5699, 5702

\g_@@_blocks_ht_dim	255, 992, 995, 996, 5690, 5693
\g_@@_blocks_seq	309, 1541, 2461, 5709, 5721, 5815
\g_@@_blocks_wd_dim	254, 980, 983, 984, 5678, 5681
\c_@@_booktabs_loaded_bool	25, 39, 1191, 2494
\l_@@_borders_clist	327, 5782, 5877, 6178, 6194, 6196, 6198, 6200, 6219, 6231
@@_cartesian_color:nn	4447, 4459, 4468, 4593
@@_cartesian_path:	4451, 4692
@@_cartesian_path:n	4499, 4634, 4692
@@_cell_begin:w	897, 1844, 1963, 2034, 2067, 2083, 2207, 2317, 2338, 2359, 2967
\l_@@_cell_box	904, 952, 954, 960, 966, 969, 973, 982, 983, 988, 989, 994, 995, 1006, 1007, 1008, 1009, 1011, 1014, 1018, 1020, 1039, 1054, 1056, 1063, 1064, 1077, 1193, 1303, 1305, 1991, 1995, 1999, 2002, 2033, 2044, 2210, 2358, 2369, 2848, 2872, 2875, 2877, 2894, 2917, 2921, 5924, 6044, 6081, 6341
@@_cell_end:	999, 1846, 1982, 2039, 2072, 2085, 2216, 2319, 2348, 2364, 2967
\l_@@_cell_space_bottom_limit_dim	502, 575, 1009, 4393
\l_@@_cell_space_top_limit_dim	501, 573, 1007, 4382
@@_cellcolor ..	1378, 4501, 4513, 4514, 4723
@@_cellcolor_tabular	1175, 4717
\g_@@_cells_seq	2638, 2639, 2640, 2642
@@_center_cell_box:	1949, 1986
@@_chessboardcolors	1386, 4506
@@_cline	155, 1241
@@_cline_i:nn	156, 157, 177, 180
@@_cline_i:w	157, 158
@@_cline_ii:w	162, 164
@@_cline_iii:w	161, 165, 166
\l_@@_code_before_bool	298, 658, 685, 1107, 1310, 1341, 1556, 2658, 2675, 2693, 2724, 2750, 2784, 2821, 3151
\g_@@_code_before_tl	1545, 1554, 1557, 3148
\l_@@_code_before_tl	297, 657, 1340, 1395, 1557
\l_@@_code_for_first_col_tl	592, 2859
\l_@@_code_for_first_row_tl	596, 916, 6013
\l_@@_code_for_last_col_tl	594, 2903
\l_@@_code_for_last_row_tl	598, 923, 6016
\l_@@_code_tl	286, 6548, 6772
\l_@@_col_max_int	321, 3274, 3285, 3353, 3413, 3430
\l_@@_col_min_int	320, 3279, 3342, 3347, 3411, 3428
\g_@@_col_total_int	250, 1022, 1269, 1349, 1360, 1432, 1617, 2289, 2290, 2743, 2744, 2824, 2828, 2833, 2834, 2893, 3027, 3029, 3041, 3598, 3616, 3676, 4255, 4742, 5354, 5364, 5398, 5485, 5827, 6051, 6597, 6646
\l_@@_col_width_dim	247, 248, 1962, 2032, 5633, 5636
@@_color_index:n	4581, 4602, 4605
\l_@@_color_int	4545, 4546, 4563, 4564, 4584, 4595
\l_@@_color_tl	329, 4578, 4579, 4589, 4592, 5545, 5620, 5622
\g_@@_colors_seq	1391, 4423, 4427, 4428, 4438
\l_@@_colors_seq	4540, 4541, 4585, 4604, 4606
@@_colortbl_like:	1173, 1260
\l_@@_colortbl_like_bool	499, 684, 1260, 1743
\c_@@_colortbl_loaded_bool	108, 112, 1210
\l_@@_cols_tl	4450, 4498, 4532, 4542, 4543, 4593, 4640, 4643
\g_@@_cols_vlism_seq	274, 1308, 1738, 1821, 6674
@@_columncolor	1385, 4462
@@_columncolor_preamble	1177, 4740
\c_@@_columncolor_regex	67, 1746
\l_@@_columns_width_dim	246, 669, 813, 2712, 2718, 5309, 5315
\g_@@_com_or_env_str	278, 281
@@_computations_for_large_nodes: ..	5425, 5438, 5443
@@_computations_for_medium_nodes: ..	5345, 5414, 5424, 5435
@@_compute_a_corner:nnnnnn	5168, 5170, 5172, 5174, 5187
@@_compute_corners:	3114, 5160
@@_construct_preamble:	1316, 1712
\l_@@_corners_cells_seq	313, 4637, 4677, 4828, 4834, 4841, 4983, 4989, 4996, 5162, 5178, 5182, 5183, 5243
\l_@@_corners_clist	518, 646, 651, 4795, 4950, 5163
@@_create_blocks_nodes:	1369, 1464
@@_create_col_nodes:	2603, 2631, 2650
@@_create_diag_nodes:	1366, 3062, 3215
@@_create_extra_nodes: ..	1462, 2460, 5335
@@_create_large_nodes:	5343, 5419
@@_create_medium_and_large_nodes: ..	5340, 5430
@@_create_medium_nodes:	5341, 5409
@@_create_nodes:	5416, 5427, 5437, 5440, 5481
@@_create_one_block_node:nnnnn	1470, 1473
@@_create_row_node:	1103, 1136, 1192
@@_cut_on_hyphen:w	358, 371, 4491, 4496, 4559, 4647, 4648, 4670, 4671, 4701, 4702, 6114, 6123, 6154, 6157, 6183, 6186, 6557, 6562, 6589, 6592
\g_@@_ddots_int	3094, 3733, 3734
@@_def_env:nnn	2940, 2951, 2952, 2953, 2954, 2955
@@_define_com:nnn	6310, 6318, 6319, 6320, 6321, 6322
@@_delimiter:nnn	2113, 2134, 2142, 2155, 2161, 2167, 6437
\l_@@_delimiters_color_tl	535, 744, 1408, 1656, 1657, 1674, 1675, 6416, 6475, 6476, 6503, 6824, 6825, 6848, 6849
\l_@@_delimiters_max_width_bool	536, 742, 1679
\g_@@_delta_x_one_dim	3096, 3736, 3746
\g_@@_delta_x_two_dim	3098, 3784, 3794
\g_@@_delta_y_one_dim	3097, 3738, 3746
\g_@@_delta_y_two_dim	3099, 3786, 3794
@@_diagbox:nn	1257, 6342
@@_dotfill:	6331

```

\@@_dotfill_i: ..... 6336, 6338
\@@_dotfill_ii: ..... 6335, 6338, 6339
\@@_dotfill_iii: ..... 6339, 6340
\l_@@_dotted_bool 346, 4850, 5005, 5285, 5292
\@@_double_int_eval:n .... 4266, 4280, 4281
\g_@@_dp_ante_last_row_dim ..... 931, 1226
\g_@@_dp_last_row_dim .....
.... 931, 932, 1229, 1230, 1304, 1305, 1650
\g_@@_dp_row_zero_dim .....
.... 951, 952, 1220, 1221, 1643, 2517, 2556
\@@_draw_Cdots:nnn ..... 3548
\@@_draw_Ddots:nnn ..... 3704
\@@_draw_Iddots:nnn ..... 3752
\@@_draw_Ldots:nnn ..... 3509
\@@_draw_Vdots:nnn ..... 3629
\@@_draw_blocks: ..... 2461, 5812
\@@_draw_dotted_lines: ..... 3113, 3182
\@@_draw_dotted_lines_i: ..... 3185, 3189
\l_@@_draw_first_bool . 336, 4050, 4074, 4085
\@@_draw_hlines: ..... 3116, 5079
\@@_draw_line: ..... 3546,
3591, 3702, 3750, 3798, 3800, 4319, 4908, 5077
\@@_draw_line_ii:nn ..... 4299, 4303
\@@_draw_line_iii:nn ..... 4306, 4310
\@@_draw_standard_dotted_line: . 3805, 3836
\@@_draw_standard_dotted_line_i: 3899, 3903
\l_@@_draw_tl ..... 325, 5776,
5780, 5865, 6096, 6102, 6104, 6106, 6143, 6144
\@@_draw_unstandard_dotted_line: 3806, 3808
\@@_draw_unstandard_dotted_line:n ...
..... 3811, 3814
\@@_draw_unstandard_dotted_line:nnn ..
..... 3816, 3821, 3835
\@@_draw_vlines: ..... 3117, 4910
\g_@@_empty_cell_bool . 306, 1013, 1023,
2882, 2929, 4001, 4017, 4033, 4057, 4080, 4091
\l_@@_end_of_row_tl .....
..... 532, 533, 586, 2623, 2624, 7077
\c_@@_endpgfortikzpicture_tl .....
..... 51, 55, 3186, 4307
\c_@@_enumitem_loaded_bool .....
..... 26, 42, 392, 712, 717, 728, 733
\@@_env: .....
... 240, 244, 937, 943, 1040, 1046, 1060,
1068, 1112, 1118, 1124, 1205, 1350, 1351,
1356, 1357, 1362, 1363, 1375, 1429, 1430,
1435, 1438, 1441, 1444, 1453, 1454, 1459,
1460, 1486, 2659, 2662, 2664, 2680, 2686,
2689, 2698, 2704, 2707, 2729, 2735, 2738,
2755, 2761, 2767, 2791, 2799, 2813, 2824,
2828, 2834, 3067, 3068, 3073, 3074, 3082,
3088, 3127, 3232, 3234, 3241, 3243, 3244,
3249, 3252, 3314, 3382, 3447, 3458, 3471,
3474, 3493, 3496, 3601, 3604, 3619, 3622,
4148, 4166, 4223, 4241, 4292, 4294, 4313,
4316, 5198, 5217, 5235, 5367, 5369, 5377,
5488, 5497, 5515, 5938, 5943, 5944, 5949,
5958, 5962, 5976, 5981, 5993, 5999, 6000,
6003, 6021, 6058, 6452, 6455, 6613, 6615,
6620, 6622, 6649, 6651, 6656, 6658, 6756, 6768
\g_@@_env_int .....
. 239, 240, 242, 1538, 1548, 1551, 1706, 5524

\@@_error:n ..... 12,
395, 420, 545, 555, 608, 738, 796, 806, 812,
821, 829, 847, 854, 862, 863, 864, 870, 875,
876, 877, 891, 893, 894, 895, 1410, 1578,
1612, 1622, 1694, 2023, 2438, 2499, 2545,
4345, 4520, 4535, 5772, 5810, 6176, 6420,
6533, 6544, 6551, 6598, 6897, 6902, 6929, 6939
\@@_error:nn ..... 13, 676, 2116,
2162, 2168, 2192, 3990, 3993, 4006, 4009,
4022, 4025, 4039, 4040, 4045, 4046, 4063,
4064, 4069, 4070, 5176, 6181, 6538, 6628, 6631
\@@_error:nnn ..... 14, 4297, 6703, 6738
\@@_error_too_much_cols: ..... 1783, 6959
\@@_everycr: ..... 1129, 1215, 1218
\@@_everycr_i: ..... 1129, 1130
\l_@@_except_borders_bool .....
..... 527, 620, 4914, 4918, 5083, 5087
\@@_exec_code_before: ..... 1310, 1389
\@@_expand_clist:N ..... 363, 1189, 1190
\@@_expand_clist:NN ..... 4640, 4641, 4693
\l_@@_exterior_arraycolsep_bool .....
..... 515, 809, 1764, 1776, 2780
\l_@@_extra_left_margin_dim .....
..... 530, 636, 1332, 2880
\l_@@_extra_right_margin_dim .....
..... 531, 637, 1573, 2925, 3679
\@@_extract_brackets ..... 5889, 6086
\@@_extract_coords_values: .... 5506, 5513
\@@_fatal:n .... 15, 270, 1529, 2094, 2123,
2608, 2612, 2614, 2647, 4106, 6964, 6967, 6970
\@@_fatal:nn ..... 16, 1835, 2311
\l_@@_fill_tl ..... 324, 5774, 5887, 5889
\l_@@_final_i_int .....
..... 3103, 3261, 3266, 3269, 3294,
3302, 3306, 3315, 3323, 3403, 3459, 3540,
3613, 3619, 3622, 4139, 4167, 4235, 4245, 4247
\l_@@_final_j_int .....
3104, 3262, 3267, 3274, 3279, 3285, 3295,
3303, 3307, 3316, 3324, 3402, 3404, 3460,
3493, 3496, 3504, 4160, 4170, 4172, 4214, 4243
\l_@@_final_open_bool .....
3106, 3268, 3272, 3275, 3282, 3288, 3292,
3308, 3537, 3572, 3577, 3588, 3652, 3662,
3667, 3688, 3725, 3773, 3907, 3922, 3953,
3954, 4137, 4161, 4173, 4212, 4236, 4248, 4289
\@@_find_extremities_of_line:nnnn ...
..... 3256, 3514, 3553, 3634, 3709, 3757
\l_@@_first_col_int .....
..... 143, 156, 349, 350, 588, 868,
903, 1360, 1432, 1626, 1756, 2653, 2673,
3039, 3598, 3616, 4099, 4661, 4754, 5354,
5364, 5398, 5446, 5485, 6291, 6297, 6303, 6646
\l_@@_first_i_tl 6585, 6590, 6609, 6640,
6649, 6651, 6706, 6713, 6715, 6797, 6808, 6812
\l_@@_first_j_tl .... 6586, 6591, 6613,
6615, 6676, 6689, 6696, 6698, 6798, 6809, 6813
\l_@@_first_row_int .. 347, 348, 589, 872,
1267, 1354, 1427, 1641, 2435, 2514, 2542,
2553, 3036, 3468, 3490, 5347, 5361, 5388,
5445, 5483, 5955, 5973, 6289, 6449, 6610, 7088
\c_@@_footnote_bool .....
1514, 1710, 6867, 6900, 6923, 6926, 6936, 6942
\c_@@_footnotehyper_bool . 6866, 6901, 6933

```

```

\@@_full_name_env: .....
    279, 6948, 6985, 6992, 7000, 7008, 7012,
    7102, 7120, 7123, 7136, 7141, 7146, 7148,
    7178, 7197, 7202, 7207, 7214, 7240, 7249, 7443
\@@_hdottedline: ..... 1247, 5272
\@@_hdottedline:n ..... 5280, 5282
\@@_hdottedline_i: ..... 5275, 5277
\@@_hline:nnnn . 4928, 5094, 5113, 5286, 6168
\@@_hline_i:nnnn ..... 4931, 4934
\@@_hline_ii:nnnn ..... 4959, 4970, 5003
\@@_hline_iii:nnnn ..... 5007, 5009
\@@_hline_iv:nnn ..... 5006, 5050
\l_@@_hlines_clist .. 342, 600, 614, 619,
    648, 1137, 1139, 1143, 1189, 3116, 5092, 5093
\l_@@_hpos_block_str .... 331, 332, 5529,
    5531, 5533, 5535, 5537, 5539, 5574, 5575,
    5577, 5628, 5637, 5650, 5661, 5712, 5735,
    5739, 5751, 5756, 5788, 5790, 5792, 5794,
    5797, 5800, 6025, 6037, 6048, 6052, 6062, 6074
\l_@@_hpos_cell_str ..... 252, 253,
    1844, 1937, 1939, 2035, 2317, 2360, 5573, 5575
\l_@@_hpos_col_str 1893, 1895, 1897, 1899,
    1924, 1936, 1940, 1942, 1950, 1951, 2018, 2184
\l_@@_hpos_of_block_cap_bool .....
    .... 333, 5795, 5798, 5801, 5952, 6022, 6059
\g_@@_ht_last_row_dim .....
    .... 933, 1227, 1228, 1302, 1303, 1649
\g_@@_ht_row_one_dim .. 959, 960, 1224, 1225
\g_@@_ht_row_zero_dim .....
    .... 953, 954, 1222, 1223, 1644, 2516, 2555
\@@_hvlines_block:nnn ..... 5855, 6150
\l_@@_hvlines_block_bool .. 337, 5784, 5851
\@@_i: ..... 5347, 5349,
    5350, 5351, 5352, 5361, 5367, 5369, 5370,
    5371, 5372, 5377, 5378, 5379, 5380, 5388,
    5391, 5393, 5394, 5395, 5447, 5449, 5452,
    5453, 5457, 5458, 5483, 5488, 5490, 5492,
    5496, 5497, 5508, 5515, 5517, 5519, 5523, 5524
\g_@@_iddots_int ..... 3095, 3781, 3782
\l_@@_in_env_bool .... 259, 419, 1529, 1530
\c_@@_in_preamble_bool . 21, 22, 23, 708, 724
\l_@@_initial_i_int ..... 3101,
    3259, 3334, 3337, 3362, 3370, 3374, 3383,
    3391, 3401, 3448, 3533, 3579, 3581, 3595,
    3601, 3604, 4138, 4139, 4149, 4217, 4227, 4229
\l_@@_initial_j_int .....
    .... 3102, 3260, 3335, 3342,
    3347, 3353, 3363, 3371, 3375, 3384, 3392,
    3402, 3404, 3449, 3471, 3474, 3482, 3669,
    3671, 3676, 4142, 4152, 4154, 4213, 4214, 4225
\l_@@_initial_open_bool .....
    .... 3105, 3336, 3340, 3343,
    3350, 3356, 3360, 3376, 3530, 3569, 3576,
    3586, 3652, 3659, 3665, 3719, 3767, 3905,
    3952, 4136, 4143, 4155, 4211, 4218, 4230, 4288
\@@_insert_tabularnotes: ..... 2466, 2469
\@@_instruction_of_type:nnn .....
    .... 1080, 3995, 4011, 4027, 4050, 4074
\c_@@_integers_alist_tl ..... 6782, 6793
\l_@@_inter_dots_dim .....
    .... 503, 504, 3110, 3910, 3917,
    3928, 3936, 3943, 3948, 3960, 3968, 5065, 5075
\g_@@_internal_code_after_tl 287, 1880,
    2112, 2133, 2141, 2154, 2160, 2166, 2227,
    3132, 3133, 5112, 5279, 5605, 5914, 6344, 6565
\@@_intersect_our_row:nnnn ..... 4623
\@@_intersect_our_row_p:nnnn ..... 4573
\@@_j: ..... 5354, 5356,
    5357, 5358, 5359, 5364, 5367, 5369, 5372,
    5374, 5375, 5377, 5380, 5382, 5383, 5398,
    5401, 5403, 5404, 5405, 5460, 5462, 5465,
    5467, 5471, 5472, 5485, 5488, 5489, 5491,
    5496, 5497, 5509, 5515, 5516, 5518, 5523, 5524
\l_@@_key_nb_rows_int .....
    .... 251, 4339, 4359, 4367, 4374
\l_@@_l_dim .....
    .... 3883, 3884, 3897, 3898, 3910, 3916,
    3927, 3935, 3943, 3948, 3960, 3961, 3968, 3969
\l_@@_large_nodes_bool 526, 627, 5339, 5343
\g_@@_last_col_found_bool ..... 357,
    1272, 1618, 1686, 2742, 2816, 2891, 3026, 6049
\l_@@_last_col_int .....
    .... 355, 356, 797, 840, 842, 855, 871,
    892, 1293, 1296, 1621, 1768, 2963, 2965,
    3027, 3029, 3639, 3674, 3992, 4008, 4046,
    4070, 5820, 5825, 5826, 5827, 5830, 5858,
    5872, 5884, 5896, 5908, 5920, 5935, 5976,
    5981, 5989, 6005, 6293, 6299, 6305, 6963, 6986
\l_@@_last_col_without_value_bool ...
    .... 354, 839, 3028, 6966
\l_@@_last_empty_column_int .....
    .... 5208, 5209, 5222, 5228, 5241
\l_@@_last_empty_row_int .....
    .... 5190, 5191, 5204, 5225
\l_@@_last_i_tl ..... 6587,
    6593, 6596, 6609, 6643, 6656, 6658, 6706, 6713
\l_@@_last_j_tl .....
    .... 6588, 6594, 6597, 6620, 6622, 6679, 6689, 6696
\l_@@_last_row_int .....
    .... 351, 352, 590, 921, 967, 1149, 1287,
    1291, 1298, 1606, 1610, 1613, 1625, 1647,
    2625, 2626, 2855, 2856, 2900, 2901, 3031,
    3519, 3558, 4024, 4040, 4064, 4760, 4768,
    5819, 5822, 5823, 5842, 5858, 5872, 5884,
    5896, 5907, 5919, 5933, 6004, 6015, 6301, 7239
\l_@@_last_row_without_value_bool ...
    .... 353, 1289, 1608, 3030
\l_@@_left_delim_dim .....
    .... 1317, 1321, 1326, 2591, 2878
\g_@@_left_delim_tl .....
    .... 1325, 1516, 1658, 1682, 1752, 2097, 2099, 5064
\l_@@_left_margin_dim .....
    .... 528, 630, 1331, 2879, 5061, 5476
\l_@@_letter_for_dotted_lines_str ...
    .... 820, 831, 832, 1816
\l_@@_letter_vlism_tl ..... 273, 607, 1819
\l_@@_light_syntax_bool 512, 584, 1334, 1568
\@@_light_syntax_i ..... 2616, 2619
\@@_line ..... 3131, 4272
\@@_line_i:nn ..... 4279, 4286
\l_@@_line_width_dim .....
    .... 330, 5786, 6097, 6135, 6146, 6152,
    6173, 6193, 6208, 6210, 6220, 6221, 6223, 6234
\@@_line_with_light_syntax:n ... 2630, 2634

```

\@@_line_with_light_syntax_i:n	\l_@@_nullify_dots_bool
..... 2629, 2635, 2636 521, 625, 3999, 4015, 4031, 4055, 4078
\@@_math_toggle_token:	\l_@@_number_of_notes_int 385, 422, 432, 442
183, 1001, 2849, 2866, 2895, 2911, 6389, 6400	\@@_old_CT@arc@
\g_@@_max_cell_width_dim	1531, 3155
..... 1010, 1011, 1540, 2717, 5302, 5328	\@@_old_cdots
\c_@@_max_l_dim	1235, 4016
3897, 3902	\@@_old_ddots
\l_@@_medium_nodes_bool 525, 626, 5337, 5996	1237, 4056
\@@_message_hdotsfor: 6972, 6985, 6992, 7000	\@@_old_dotfill
\@@_msg_new:nn	6330, 6333, 6341
17, 6881,	\@@_old_dotfill:
6905, 6914, 6944, 6977, 6982, 6989, 6997,	1255
7005, 7010, 7016, 7021, 7026, 7031, 7036,	\l_@@_old_iRow_int
7041, 7046, 7051, 7057, 7063, 7068, 7074,	288, 1283, 3202
7080, 7085, 7092, 7099, 7105, 7114, 7119,	\@@_old_ialign:
7121, 7127, 7144, 7151, 7156, 7163, 7170,	1102, 1231, 5814
7176, 7183, 7190, 7195, 7205, 7211, 7218,	\@@_old_iddots
7225, 7231, 7237, 7245, 7247, 7253, 7258, 7555	1238, 4079
\@@_msg_new:nnn 18, 6868, 7133, 7264, 7274,	\l_@@_old_jCol_int
7289, 7310, 7328, 7345, 7388, 7440, 7491, 7541	289, 1285, 3203
\@@_msg_redirect_name:nn	\@@_old_ldots
..... 19, 815, 1698, 5833, 5836	1234, 4000
\@@_multicolumn:nnn	\@@_old_multicolumn
1262, 2257	1264, 4094
\g_@@_multicolumn_cells_seq	\@@_old_pgfpaintanchor
..... 316, 1265, 1312, 2272,	191, 6774, 6778
3052, 3056, 3057, 5372, 5380, 5502, 5960, 5978	\@@_old_pgfulfil@check@rerun
\g_@@_multicolumn_sizes_seq	101, 105
..... 317, 1266, 1313, 2274, 3058, 3059, 5503	\@@_old_vdots
\g_@@_name_env_str	1236, 4032
277, 282,	\@@_open_x_final_dim:
283, 1523, 1524, 2646, 2935, 2936, 2944, 3487, 3539, 3573, 3727, 3776
2945, 2974, 2987, 3006, 3016, 3153, 6314, 6961	\@@_open_x_initial_dim:
\l_@@_name_str 3465, 3532, 3570, 3722, 3770
... 524, 678, 939, 942, 1042, 1045, 1120,	\@@_open_y_final_dim:
1123, 2663, 2664, 2688, 2689, 2706, 2707,	3611, 3663, 3775
2737, 2738, 2763, 2766, 2809, 2812, 2830,	\@@_open_y_initial_dim:
2833, 3076, 3081, 3087, 3233, 3234, 3245, 3593, 3660, 3721, 3769
3248, 3251, 5493, 5496, 5520, 5523, 5945, 5948	\l_@@_parallelize_diags_bool
\g_@@_names_seq 516, 517, 622, 3092, 3731, 3779
258, 675, 677, 7553	\@@_patch_for_revtext:
\l_@@_nb_cols_int	1493, 1512
..... 6267, 6278, 6286, 6292, 6298, 6304	\@@_patch_m_preamble:n
\l_@@_nb_rows_int 2266, 2293, 2326, 2331, 2392
6266, 6277, 6295	\@@_patch_m_preamble_i:n
\@@_newcolumnntype 1156, 1717, 1718, 1719, 2993 2297, 2298, 2299, 2313
\@@_node_for_cell:	\@@_patch_m_preamble_ii:nn
..... 1019, 1026, 1400, 2876, 2926 2300, 2301, 2302, 2323
\@@_node_for_multicolumn:nn	\@@_patch_m_preamble_iii:n
5504, 5511	2303, 2328
\@@_node_left:nn	\@@_patch_m_preamble_iv:nnn
6755, 6756, 6816 2304, 2305, 2306, 2333
\@@_node_position:	\@@_patch_m_preamble_ix:n
1356, 1358, 1362, 1364, 1429, 1431, 1439, 1445	2377, 2395
\@@_node_position_i:	\@@_patch_m_preamble_v:nnnn 2307, 2308, 2353
1442, 1446	\@@_patch_m_preamble_x:n
\@@_node_right:nnnn 2321, 2351, 2372, 2374, 2398
6765, 6767, 6840	\@@_patch_node_for_cell:
\g_@@_not_empty_cell_bool	1052, 1400
..... 296, 1017, 1024, 2982	\@@_patch_node_for_cell:n 1050, 1076, 1079
\@@_not_in_exterior:nnnnn	\@@_patch_preamble:n
4615 1740, 1786, 1825, 1833, 1854, 1887,
\@@_not_in_exterior_p:nnnnn	2101, 2117, 2119, 2135, 2143, 2169, 2229, 2249
4551	\@@_patch_preamble_i:n 1790, 1791, 1792, 1840
\l_@@_notes_above_space_dim	\@@_patch_preamble_ii:nn
519, 520 1793, 1794, 1795, 1851
\l_@@_notes_bottomrule_bool	\@@_patch_preamble_iii:n . 1796, 1856, 1864
..... 696, 858, 886, 2492	\@@_patch_preamble_iii_i:n
\l_@@_notes_code_after_tl	1859, 1861
694, 2501	\@@_patch_preamble_iv:n
\l_@@_notes_code_before_tl 1797, 1798, 1799, 1909
692, 2473	\@@_patch_preamble_iv_i:n
\@@_notes_label_in_list:n 388, 407, 415, 704	1912, 1914
\@@_notes_label_in_tabular:n . 387, 428, 701	\@@_patch_preamble_iv_ii:w 1917, 1918, 1920
\l_@@_notes_para_bool .. 690, 856, 884, 2477	\@@_patch_preamble_iv_iii:nn ...
\@@_notes_style:n	1921, 1922
..... 386, 389, 407, 415, 431, 436, 698	\@@_patch_preamble_iv_iv:nn
 1926, 1928, 2021, 2024, 2199
	\@@_patch_preamble_iv_v:nnnnnnn 1932, 1957
	\@@_patch_preamble_ix:nn
 1808, 1809, 1810, 2121
	\@@_patch_preamble_ix_i:nnn
	2125, 2147
	\@@_patch_preamble_v:n ...
	1800, 1801, 2007
	\@@_patch_preamble_v_i:w . 2010, 2011, 2013

<code>\@@_patch_preamble_v_ii:nn</code> 2014, 2015	6018, 6119, 6121, 6128, 6130, 6207, 6209,
<code>\@@_patch_preamble_vi:nnnn</code>	1802, 1803, 2027	6211, 6218, 6222, 6224, 6368, 6370, 6373,
<code>\@@_patch_preamble_vii:n</code> 1804, 2050	6375, 6442, 6444, 6640, 6643, 6681, 6698, 6715
<code>\@@_patch_preamble_vii_i:w</code>	2053, 2054, 2056	<code>\l_@@_radius_dim</code>
<code>\@@_patch_preamble_vii_ii:n</code>	2057, 2062, 2079	507, 508, 2226, 3109, 3544, 3545, 3977, 5274
<code>\@@_patch_preamble_viii:nn</code>	<code>\l_@@_real_left_delim_dim</code>
..... 1805, 1806, 1807, 2092		2562, 2577, 2592
<code>\@@_patch_preamble_viii_i:nn</code>	<code>\l_@@_real_right_delim_dim</code>
..... 2105, 2108, 2110		2563, 2589, 2595
<code>\@@_patch_preamble_x:n</code>	... 1811, 1812, 2172	<code>\@@_recreate_cell_nodes:</code>
<code>\@@_patch_preamble_x_i:w</code>	2175, 2176, 2178 1367, 1425
<code>\@@_patch_preamble_x_ii:n</code> 2179, 2182	<code>\g_@@_recreate_cell_nodes_bool</code>
<code>\@@_patch_preamble_xi:n</code> 523, 1200, 1367, 1392, 1399, 1404
1849, 1955, 2048, 2075, 2088, 2220, 2231, 2255		<code>\@@_rectanglecolor</code>
<code>\@@_patch_preamble_xii:n</code> 1817, 2223 1379, 4354, 4471, 4504, 4524, 4734
<code>\@@_patch_preamble_xiii:n</code> 2234, 2252	<code>\@@_rectanglecolor:nnn</code>
<code>\@@_pgf_rect_node:nnn</code> 475, 1443, 5998	... 4477, 4486, 4489
<code>\@@_pgf_rect_node:nnnn</code>	<code>\@@_renew_NC@rewrite@S:</code>
.... 450, 1485, 5487, 5514, 5937, 5992, 6742	 202, 204, 1271
<code>\c_@@_pgfortikzpicture_tl</code>	50, 54, 3184, 4305	<code>\@@_renew_dots:</code>
<code>\@@_pgfpointanchor:n</code> 6770, 6775 1163, 1261
<code>\@@_pgfpointanchor_i:nn</code> 6778, 6780	<code>\l_@@_renew_dots_bool</code>
<code>\@@_pgfpointanchor_ii:w</code> 6781, 6789 623, 805, 1261, 6889, 6896
<code>\@@_pgfpointanchor_iii:w</code> 6802, 6804	<code>\@@_renew_matrix:</code>
<code>\@@_picture_position:</code>	800, 804, 6246, 6891, 6895
..... 1351, 1358, 1364, 1431, 1445, 1446		<code>\l_@@_respect_blocks_bool</code>
<code>\g_@@_pos_of_blocks_seq</code>	4530, 4547, 4570
310, 1311, 1469, 1542, 2275, 3044, 3048,		<code>\@@_restore_iRow_jCol:</code>
3049, 3161, 4549, 4789, 4944, 5255, 5862, 6354	 3154, 3200
<code>\g_@@_pos_of_stroken_blocks_seq</code>	<code>\c_@@_revtex_bool</code>
..... 312, 1543, 4793, 4948, 5874	 58, 60, 63, 65, 1512
<code>\g_@@_pos_of_xdots_seq</code>	<code>\l_@@_right_delim_dim</code>
..... 311, 1544, 3399, 4791, 4946	 1318, 1322, 1328, 2594, 2923
<code>\g_@@_post_action_cell_tl</code>	<code>\g_@@_right_delim_tl</code>
..... 899, 1003, 1988, 2209, 4380, 4391		.. 1327, 1517, 1676,
<code>\@@_pre_array:</code> 1281, 1342, 1565	1682, 1753, 2100, 2129, 2130, 2151, 2156, 5074
<code>\@@_pre_array_i:w</code> 1338, 1565	<code>\l_@@_right_margin_dim</code>
<code>\@@_pre_array_ii:</code> 1179, 1314 529, 632, 1572, 2924, 3678, 5071, 5479
<code>\@@_pre_code_before:</code> 1344, 1421	<code>\@@_rotate:</code>
<code>\c_@@_preamble_first_col_tl</code> 1757, 2841 1253, 4265
<code>\c_@@_preamble_last_col_tl</code> 1769, 2886	<code>\g_@@_rotate_bool</code>
<code>\g_@@_preamble_tl</code> 264, 976, 1004, 1979, 2042, 2367, 2868,
.... 1518, 1720, 1724, 1728, 1734, 1748,		2913, 4265, 5628, 5669, 5674, 5734, 5750, 5925
1757, 1766, 1769, 1778, 1782, 1823, 1832,		<code>\@@_rotate_cell_box:</code>
1842, 1853, 1866, 1959, 2029, 2064, 2081,	 964, 1004, 2042, 2367, 2868, 2913, 5925
2104, 2132, 2140, 2153, 2159, 2204, 2225,		<code>\l_@@_rounded_corners_dim</code>
2238, 2245, 2254, 2265, 2267, 2315, 2325,		328, 5778, 5897, 6111, 6112, 6147, 6175, 6232
2330, 2335, 2355, 2381, 2388, 2397, 2600, 2627		<code>\@@_roundedrectanglecolor</code>
<code>\@@_pred:n</code> 144, 1380, 4480
182, 2965, 4829, 4842, 4984, 4997, 6163, 6168		<code>\l_@@_row_max_int</code>
<code>\@@_provide_pgfsyspdfmark:</code>	... 68, 77, 1513 319, 3269, 3412, 3429
<code>\@@_put_box_in_flow:</code> 1684, 2400, 2593	<code>\l_@@_row_min_int</code>
<code>\@@_put_box_in_flow_bis:nn</code> 1681, 2560 318, 3337, 3410, 3427
<code>\@@_put_box_in_flow_i:</code> 2406, 2408	<code>\g_@@_row_of_col_done_bool</code>
<code>\@@_qpoint:n</code> 295, 1134, 1522, 2672
243, 1477, 1479, 1481, 1483, 2411, 2413,		<code>\g_@@_row_style_tl</code>
2425, 2441, 2508, 2510, 2526, 2537, 2548,	 299, 911, 1546, 1971, 4372, 4373,
3221, 3223, 3225, 3227, 3237, 3239, 3482,		4375, 4378, 4389, 4400, 4405, 4416, 4417, 5623
3504, 3533, 3540, 3579, 3581, 3595, 3613,		<code>\g_@@_row_total_int</code>
3669, 3671, 4313, 4316, 4660, 4664, 4680,	 249, 1268,
4682, 4858, 4860, 4862, 4901, 4904, 4906,		1348, 1354, 1427, 1624, 2436, 2543, 3031,
5013, 5015, 5017, 5054, 5057, 5067, 5393,		3038, 3468, 3490, 4180, 5347, 5361, 5388,
5403, 5929, 5931, 5933, 5935, 5969, 5989,		5483, 5842, 5955, 5973, 6449, 6596, 6610, 7089
		<code>\@@_rowcolor</code>
	 1381, 4363, 4453
		<code>\@@_rowcolor_tabular</code>
	 1176, 4728
		<code>\@@_rowcolors</code>
	 1382, 4608
		<code>\@@_rowcolors_i:nnnnn</code>
	 4574, 4610
		<code>\l_@@_rowcolors_restart_bool</code>
		... 4533, 4562
		<code>\@@_rowlistcolors</code>
	 1383, 4537, 4609
		<code>\g_@@_rows_seq</code>
		.. 2622, 2624, 2626, 2628, 2630
		<code>\l_@@_rows_tl</code>
	 4449, 4497, 4576, 4593, 4641, 4666
		<code>\l_@@_rules_color_tl</code>
	 290, 559, 1563, 1564, 6671, 6672
		<code>\@@_set_CT@arc@:</code>
	 185, 1564, 6672
		<code>\@@_set_CT@arc@_i:</code>
	 186, 187
		<code>\@@_set_CT@arc@_ii:</code>
	 186, 189
		<code>\@@_set_final_coords:</code>
	 3438, 3463

<code>\@@_set_final_coords_from_anchor:n</code> ...	<code>\@@_test_hline_in_stroken_block:nnnn</code> .
... 3454, 3543, 3574, 3655, 3664, 3730, 3778 4949, 5138
<code>\@@_set_initial_coords:</code> 3433, 3452	<code>\@@_test_if_cell_in_a_block:nn</code>
<code>\@@_set_initial_coords_from_anchor:n</code> 5194, 5212, 5230, 5250
... 3443, 3536, 3571, 3654, 3661, 3724, 3772	<code>\@@_test_if_cell_in_block:nnnnnnn</code> ...
<code>\@@_set_size:n</code> 6264, 6279 5256, 5258
<code>\c_@@_siunitx_loaded_bool</code> 192, 196, 201, 219	<code>\@@_test_if_math_mode:</code> 267, 1528, 2946
<code>\c_@@_size_seq</code>	<code>\@@_test_in_corner_h:</code> 4950, 4978
... 1291, 1296, 1346, 1347, 1348, 1349, 3034	<code>\@@_test_in_corner_v:</code> 4795, 4823
<code>\l_@@_small_bool</code> 798, 845, 851,	<code>\@@_test_vline_in_block:nnnnn</code>
873, 908, 1195, 2094, 2123, 2850, 2896, 3107 4790, 4792, 5127
<code>\@@_standard_cline</code> 140, 1240	<code>\@@_test_vline_in_stroken_block:nnnn</code> .
<code>\@@_standard_cline:w</code> 140, 141 4794, 5149
<code>\l_@@_standard_cline_bool</code> .. 500, 571, 1239	<code>\l_@@_the_array_box</code>
<code>\c_@@_standard_tl</code> 510, 511, 3804	1315, 1330, 1590, 1598, 2453, 2454, 2456, 2459
<code>\g_@@_static_num_of_col_int</code>	<code>\c_@@_tikz_loaded_bool</code>
... 323, 1693, 1741, 5830, 7001, 7013, 7198 28, 49, 1370, 3119, 5770
<code>\l_@@_stop_loop_bool</code> 3263, 3264,	<code>\l_@@_tikz_seq</code> 326, 5771, 5900, 5909
3296, 3309, 3318, 3331, 3332, 3364, 3377, 3386	<code>\g_@@_total_X_weight_int</code>
<code>\@@_store_in_tmpb_tl</code> 6089, 6091 291, 1188, 1577, 1580, 1599, 2196
<code>\@@_stroke_block:nnn</code> 5869, 6093	<code>\l_@@_type_of_col_tl</code> 843,
<code>\@@_stroke_borders_block:nnn</code> ... 5881, 6171	844, 2975, 2977, 6271, 6272, 6273, 6281, 6286
<code>\@@_stroke_horizontal:n</code> .. 6199, 6201, 6216	<code>\c_@@_types_of_matrix_seq</code>
<code>\@@_stroke_vertical:n</code> 6195, 6197, 6205 6951, 6952, 6957, 6961
<code>\@@_sub_matrix:nnnnnnn</code> 6577, 6582	<code>\@@_update_for_first_and_last_row:</code> ..
<code>\@@_sub_matrix_i:nnnn</code> 6632, 6638 947, 1012, 1300, 2870, 2915
<code>\l_@@_submatrix_extra_height_dim</code>	<code>\@@_use_arraybox_with_notes:</code> ... 1638, 2521
..... 339, 6495, 6666	<code>\@@_use_arraybox_with_notes_b:</code> . 1635, 2505
<code>\l_@@_submatrix_hlines_clist</code>	<code>\@@_use_arraybox_with_notes_c:</code>
..... 344, 6507, 6525, 6705, 6707 1636, 1667, 2449, 2519, 2558
<code>\l_@@_submatrix_left_xshift_dim</code>	<code>\c_@@_varwidth_loaded_bool</code> ... 29, 33, 2020
..... 340, 6497, 6718, 6751	<code>\@@_vdottedline:n</code> 2228, 5289
<code>\l_@@_submatrix_name_str</code>	<code>\@@_vline:nnnn</code> . 1882, 4770, 4925, 5293, 6163
6540, 6600, 6740, 6742, 6754, 6756, 6764, 6768	<code>\@@_vline_i:nnnn</code> 4775, 4779
<code>\g_@@_submatrix_names_seq</code>	<code>\@@_vline_ii:nnnn</code> 4804, 4815, 4848
..... 314, 3135, 6537, 6541, 7142	<code>\@@_vline_iii:nnnn</code> 4852, 4854
<code>\l_@@_submatrix_right_xshift_dim</code>	<code>\@@_vline_iv:nnn</code> 4851, 4897
..... 341, 6499, 6727, 6761	<code>\l_@@_vlines_clist</code> 343, 601, 613,
<code>\g_@@_submatrix_seq</code> ... 322, 1309, 3414, 6563	618, 647, 1190, 1726, 1732, 1763, 1775,
<code>\l_@@_submatrix_slim_bool</code> 6505, 6608, 7110	2236, 2243, 2379, 2386, 2779, 3117, 4923, 4924
<code>\l_@@_submatrix_vlines_clist</code>	<code>\l_@@_vpos_col_str</code>
..... 345, 6509, 6527, 6688, 6690	1901, 1904, 1906, 1911, 1933, 1949, 2017, 2185
<code>\@@_succ:n</code> 177, 181,	<code>\l_@@_vpos_of_block_tl</code> .. 334, 335, 5541,
1112, 1118, 1123, 1124, 1144, 1481, 1483,	5543, 5635, 5649, 5660, 5738, 5755, 5803, 5805
1883, 2113, 2243, 2273, 2386, 2413, 2755,	<code>\@@_w:</code> 1717, 1802, 2307
2761, 2766, 2767, 2791, 2799, 2812, 2813,	<code>\l_@@_weight_int</code>
2824, 2828, 2833, 2834, 3504, 3581, 3671,	2181, 2186, 2187, 2190, 2193, 2194, 2196, 2200
4619, 4664, 4680, 4825, 4862, 4906, 4919,	<code>\l_@@_width_dim</code> 257,
4980, 5017, 5067, 5088, 5113, 5260, 5262,	793, 881, 1590, 1598, 2985, 2986, 3007, 3008
5264, 5266, 5453, 5457, 5467, 5471, 5933,	<code>\g_@@_width_first_col_dim</code>
5935, 5989, 6128, 6130, 6242, 6373, 6375, 6444 308, 1521, 1629, 2667, 2871, 2872
<code>\l_@@_suffix_tl</code> 5415, 5426,	<code>\g_@@_width_last_col_dim</code>
5436, 5439, 5488, 5496, 5497, 5515, 5523, 5524 307, 1520, 1688, 2820, 2916, 2917
<code>\c_@@_table_collect_begin_tl</code> 230, 232, 2083	<code>\l_@@_width_used_bool</code> 315, 882, 1575
<code>\c_@@_table_print_tl</code> 233, 234, 2085	<code>\l_@@_x_final_dim</code> 302, 3440, 3489, 3498,
<code>\l_@@_tabular_width_dim</code>	3499, 3502, 3505, 3506, 3657, 3673, 3681,
..... 262, 1096, 1098, 1780, 3017	3685, 3689, 3691, 3696, 3698, 3728, 3737,
<code>\l_@@_tabularnote_tl</code> 384, 860, 888, 2465, 2474	3745, 3785, 3793, 3832, 3847, 3856, 3890,
<code>\g_@@_tabularnotes_seq</code>	3942, 3958, 4317, 4903, 5068, 5071, 5073,
..... 383, 423, 2480, 2486, 2502	5075, 6607, 6623, 6624, 6630, 6727, 6744, 6761
<code>\c_@@_tabularx_loaded_bool</code> ... 27, 45, 3004	<code>\l_@@_x_initial_dim</code> 300,
<code>\@@_test_hline_in_block:nnnnn</code>	3435, 3467, 3476, 3477, 3480, 3483, 3484,
..... 4945, 4947, 5116	3657, 3672, 3673, 3680, 3685, 3689, 3691,

3693, 3696, 3698, 3737, 3745, 3785, 3793,
 3829, 3846, 3856, 3890, 3942, 3956, 3958,
 3976, 3978, 4314, 4902, 5058, 5061, 5063,
 5065, 6606, 6616, 6617, 6627, 6718, 6743, 6751
 $\backslash l_{@@_xdots_color_tl}$ 534, 548, 3523, 3562,
 3643, 3644, 3712, 3760, 3812, 4184, 4259, 4276
 $\backslash l_{@@_xdots_down_tl}$... 552, 3819, 3840, 3875
 $\backslash l_{@@_xdots_line_style_tl}$
 509, 511, 544, 3804, 3812
 $\backslash l_{@@_xdots_shorten_dim}$ 505,
 506, 550, 3111, 3826, 3827, 3916, 3927, 3935
 $\backslash l_{@@_xdots_up_tl}$ 553, 3818, 3839, 3865
 $\backslash l_{@@_y_final_dim}$
 303, 3441, 3541, 3545, 3583, 3587,
 3589, 3614, 3624, 3625, 3739, 3742, 3787,
 3790, 3832, 3847, 3855, 3892, 3947, 3966,
 4318, 4907, 5056, 6445, 6467, 6482, 6644,
 6659, 6660, 6665, 6683, 6700, 6744, 6752, 6762
 $\backslash l_{@@_y_initial_dim}$ 301, 3436, 3534,
 3544, 3582, 3583, 3587, 3589, 3596, 3606,
 3607, 3739, 3744, 3787, 3792, 3829, 3846,
 3855, 3892, 3947, 3964, 3966, 3976, 3979,
 4315, 4905, 5055, 6443, 6467, 6482, 6641,
 6652, 6653, 6665, 6682, 6699, 6743, 6752, 6762
 $\backslash \{$.. 2613, 2635, 6293, 6299, 6305, 6423, 6870,
 6871, 6911, 6920, 6948, 7013, 7018, 7023,
 7028, 7033, 7071, 7076, 7082, 7089, 7095,
 7096, 7111, 7116, 7124, 7130, 7136, 7137,
 7148, 7160, 7167, 7179, 7186, 7193, 7202,
 7208, 7215, 7222, 7228, 7234, 7246, 7250,
 7255, 7261, 7267, 7276, 7277, 7291, 7292,
 7308, 7312, 7313, 7331, 7332, 7348, 7349,
 7391, 7392, 7443, 7444, 7494, 7495, 7548, 7549
 $\backslash \{$ 283, 1807, 2114,
 2139, 2953, 6322, 6723, 7129, 7215, 7391, 7494
 $\backslash \}$ 283, 1810, 2114,
 2124, 2953, 6322, 6732, 7129, 7215, 7391, 7494
 $\backslash |$ 2955, 6321
 $\backslash \sqcup$.. 6947, 6975, 6985, 6992, 7000, 7001, 7012,
 7013, 7070, 7088, 7089, 7102, 7108, 7112,
 7120, 7123, 7135, 7141, 7153, 7197, 7198,
 7199, 7207, 7213, 7220, 7233, 7240, 7241, 7249

A

$\backslash A$ 6535
 $\backslash aboverulesep$ 2496
 $\backslash addtocounter$ 440
 $\backslash alph$ 386
 $\backslash anchor$ 3212, 3213
 $\backslash array$ 1501
 $\backslash arraybackslash$ 1972, 2211, 2341
 $\backslash arraycolor$ 1384, 6947
 $\backslash arraycolsep$
 .. 631, 633, 635, 1095, 1198, 1321, 1322,
 1666, 1670, 2454, 2790, 2794, 2804, 5063, 5073
 $\backslash arrayrulecolor$ 115
 $\backslash arrayrulewidth$ 148, 153, 173, 561, 938, 1111,
 1113, 1119, 1150, 1729, 1735, 1824, 1874,
 2239, 2246, 2382, 2389, 2513, 2552, 2679,
 2681, 2687, 2697, 2699, 2705, 2728, 2730,
 2736, 2754, 2756, 2762, 2788, 2792, 2804,
 2807, 4662, 4663, 4665, 4681, 4683, 4873,

4874, 4876, 4887, 4893, 5029, 5040, 5046,
 5109, 5328, 6097, 6152, 6173, 6467, 6666, 6670
 $\backslash arraystretch$
 1197, 3597, 3615, 5625, 5731, 5747, 6642, 6645
 $\backslash AtBeginDocument$ 23, 30, 93, 109, 193,
 199, 215, 390, 504, 506, 508, 520, 710, 726,
 2058, 3180, 3983, 4113, 4190, 4268, 4301, 6569
 $\backslash AtBeginEnvironment$ 139, 1263
 $\backslash AutoNiceMatrix$ 6323
 $\backslash AutoNiceMatrixWithDelims$... 6275, 6315, 6327

B

$\backslash baselineskip$ 118, 124, 2000
 $\backslash begingroup$ 2260
 $\backslash bfseries$ 4409, 4412
 $\backslash bgroup$ 1515
 $\backslash Block$ 1252, 7166, 7173, 7220, 7267
 $\backslash BNiceMatrix$ 6261
 $\backslash bNiceMatrix$ 6258
 $\backslash Body$ 1338
 $\backslash boldmath$ 4409, 4412
 bool commands:
 $\backslash bool_do_until:Nn$ 3264, 3332
 $\backslash bool_gset_false:N$ 976,
 1023, 1024, 1133, 1272, 1392, 1522, 1547,
 1725, 2882, 2929, 5125, 5136, 5147, 5158, 5674
 $\backslash bool_gset_true:N$
 1550, 1886, 2672, 2846, 2891, 2982, 4001,
 4017, 4033, 4057, 4080, 4091, 4265, 4788, 4943
 $\backslash bool_if:N\TF$. 184, 712, 717, 728, 733, 905,
 908, 1004, 1107, 1134, 1191, 1195, 1200,
 1260, 1261, 1290, 1295, 1310, 1367, 1370,
 1394, 1397, 1399, 1419, 1512, 1514, 1529,
 1539, 1575, 1608, 1686, 1691, 1710, 1715,
 1743, 1755, 1979, 2042, 2094, 2123, 2367,
 2492, 2658, 2675, 2693, 2711, 2724, 2750,
 2784, 2816, 2821, 2850, 2868, 2896, 2913,
 3004, 3026, 3028, 3030, 3092, 3107, 3119,
 3136, 3139, 3586, 3588, 3731, 3779, 3999,
 4015, 4031, 4055, 4078, 4403, 4547, 4570,
 5062, 5072, 5203, 5221, 5239, 5311, 5321,
 5343, 5628, 5669, 5734, 5750, 5925, 5952,
 5996, 6022, 6059, 6334, 6923, 6933, 6966, 7110
 $\backslash bool_if:n\TF$
 201, 392, 419, 1082, 1618, 3419,
 4290, 4625, 4914, 4918, 5083, 5087, 5337,
 5586, 6049, 6446, 6456, 6458, 6471, 6477, 6486
 $\backslash bool_lazy_all:n\TF$ 1759,
 1771, 2775, 4864, 5019, 5118, 5129, 5140, 5151
 $\backslash bool_lazy_and:nn\TF$ 218,
 1828, 2451, 2714, 2789, 2793, 2801, 2851,
 3575, 3838, 4097, 4636, 5631, 6115, 6692, 6709
 $\backslash bool_lazy_or:nn\TF$
 541, 1016, 1074, 1751, 2434, 2463,
 2541, 2899, 3652, 3896, 4617, 4649, 4653,
 4703, 4707, 5195, 5213, 5231, 5561, 5566, 6595
 $\backslash bool_lazy_or_p:nn$ 2854
 $\backslash bool_not_p:n$
 ... 1762, 1764, 1774, 1776, 2716, 2778, 2780
 $\backslash bool_set:Nn$ 3656
 $\backslash c_false_bool$
 2134, 2142, 2155, 2161, 2167, 3995, 4011, 4027

```

\g_tmpa_bool ..... 119, 125, 188,
    .... 4788, 4796, 4830, 4838, 4843, 4943,
    4951, 4985, 4993, 4998, 5125, 5136, 5147, 5158
\g_tmpb_bool ..... 1725, 1755, 1886
\l_tmpb_bool ... 5200, 5214, 5232, 5254, 5267
\c_true_bool ..... 2113
box commands:
\box_clear_new:N ..... 1193, 1315
\box_dp:N .....
    .. 932, 952, 989, 1009, 1064, 1221, 1230,
    1305, 2346, 2403, 2571, 2584, 3615, 5704, 6645
\box_gclear_new:N ..... 5615
\box_grotate:Nn ..... 5671
\box_ht:N .....
    .... 933, 954, 960, 972, 995, 1007, 1056,
    1223, 1225, 1228, 1303, 1967, 1991, 1993,
    1999, 2342, 2402, 2571, 2584, 3597, 5695, 6642
\box_move_down:n ..... 1064, 1997
\box_move_up:n .....
    .... 84, 86, 88, 1056, 2446, 2519, 2558
\box_rotate:N ..... 966
\box_set_dp:N ..... 988, 1008, 2403
\box_set_ht:N ..... 994, 1006, 2402
\box_set_wd:N ..... 982, 2453
\box_use:N ..... 443, 973, 1063, 2002
\box_use_drop:N .....
    .... 1014, 1020, 1039, 2044, 2369,
    2405, 2446, 2447, 2459, 2877, 5714, 6044, 6081
\box_wd:N ..... 444, 983, 1011, 1018,
    1077, 1326, 1328, 1590, 1598, 2454, 2456,
    2578, 2590, 2872, 2875, 2917, 2921, 5683, 6341
\l_tmpa_box .....
    .. 426, 443, 444, 1325, 1326, 1327, 1328,
    1653, 2402, 2403, 2405, 2446, 2447, 2571, 2584
\l_tmpb_box ..... 2564, 2578, 2579, 2590
\color ..... 119, 125, 188,
    190, 910, 1657, 1675, 3517, 3520, 3523,
    3556, 3559, 3562, 3637, 3640, 3644, 3712,
    3760, 4178, 4181, 4184, 4253, 4256, 4259,
    4276, 4401, 4440, 5622, 5780, 6476, 6825, 6849
\colorlet ... 275, 276, 917, 924, 1187, 2860, 2904
\columncolor ..... 1177, 1385, 3145, 4746
\cr ..... 152, 178, 2839
\crrc ..... 2652
cs commands:
\cs_generate_variant:Nn .....
    .... 66, 180, 3835, 4432, 4433, 5333, 5334
\cs_gset:Npn ..... 119, 125, 5326
\cs_gset_eq:NN .... 77, 235, 1214, 1531, 3155
\cs_if_exist:NTF ..... 65, 139, 1283,
    1285, 1434, 1532, 1535, 1719, 2060, 3202,
    3203, 3299, 3312, 3367, 3380, 3470, 3492,
    3600, 3618, 4146, 4164, 4221, 4239, 5313,
    5366, 5957, 5975, 6451, 6612, 6619, 6648, 6655
\cs_if_exist_p:N .....
    .... 220, 542, 4867, 5022, 5197, 5216, 5234
\cs_if_free:NTF .....
    .... 73, 2995, 3512, 3551, 3632, 3707, 3755
\cs_if_free_p:N ..... 4292, 4294
\cs_new_protected:Npx ..... 3182, 4303
\cs_set:Nn ..... 698, 701, 704
\cs_set:Npn ..... 115,
    116, 121, 122, 127, 140, 141, 155, 157, 158,
    164, 166, 188, 190, 389, 1158, 1506, 1507,
    2261, 2284, 2997, 3258, 3320, 3388, 4188,
    4263, 5097, 5098, 5104, 5105, 5625, 5731, 5747
\cs_set_nopar:Npn 1097, 1197, 1208, 5508, 5509
\cs_set_nopar:Npx ..... 1098
\cs_set_protected:Npn ..... 6312
\cs_set_protected_nopar:Npn .... 5603, 5912

```


<code>\doublerulesep</code>	1875, 4876, 4888, 5029, 5041, 5110	<code>\fp_to_dim:n</code>	3886
<code>\doublerulesepcolor</code>	121	<code>\futurelet</code>	134
<code>\draw</code>	3823		
E			
<code>\egroup</code>	1507, 1701	<code>\globaldefs</code>	1697, 5835
<code>\else</code>	2261	group commands:	
else commands:		<code>\group_insert_after:N</code>	6335, 6336, 6338, 6339
<code>\else:</code>	269, 4411		
<code>\endarray</code>	1505, 1507, 2604, 2632	H	
<code>\endBNiceMatrix</code>	6262	<code>\halign</code>	1232
<code>\endbNiceMatrix</code>	6259	<code>\hbox</code>	1105, 1662, 2457, 2519, 2558, 2677, 2695, 2722, 2726, 2752, 2786
<code>\endgroup</code>	2269	hbox commands:	
<code>\endNiceArray</code>	2992, 3013, 3022	<code>\hbox:n</code>	84, 86, 89
<code>\endNiceArrayWithDelims</code>	2939, 2949	<code>\hbox_gset:Nn</code>	5617
<code>\endpgfscope</code>	3880, 6394	<code>\hbox_overlap_left:n</code>	1057, 1065, 2656, 2873
<code>\endpNiceMatrix</code>	6250	<code>\hbox_overlap_right:n</code>	443, 2818, 2919
<code>\endsavenotes</code>	1710	<code>\hbox_set:Nn</code>	426, 1054, 1325, 1327, 1653, 1995, 2210, 2564, 2579, 5924
<code>\endtabular</code>	1507	<code>\hbox_set:Nw</code>	904, 1330, 2033, 2358, 2848, 2894
<code>\endtabularnotes</code>	2487	<code>\hbox_set_end:</code>	1002, 1574, 2041, 2366, 2867, 2912
<code>\endVNiceMatrix</code>	6256	<code>\hbox_to_wd:nn</code>	468, 493
<code>\endvNiceMatrix</code>	6253	<code>\Hdotsfor</code>	1250, 6975, 7153
<code>\enskip</code>	2104, 2132, 2140, 2153, 2159	<code>\hdotsfor</code>	1171
<code>\ensuremath</code>	4000, 4016, 4032, 4056, 4079	<code>\hdottedline</code>	1247
<code>\everycr</code>	151, 178, 1218	<code>\heavyrulewidth</code>	2497
<code>\everypar</code>	1965, 1968	<code>\hfil</code>	1803, 2308
exp commands:		<code>\hfill</code>	148, 173
<code>\exp_after:wN</code>	208, 1564, 1658, 1676, 1740, 2266, 6672	<code>\Hline</code>	1248, 5100
<code>\exp_args:Nnc</code>	5315	<code>\hline</code>	127
<code>\exp_args:Nne</code>	2977	<code>\hrule</code>	131, 148, 173, 1150, 2497, 6481, 6830, 6854
<code>\exp_args:NNV</code>	2624, 3987, 4003, 4019, 4035, 4059, 4117, 4194, 4272, 6573	<code>\hsize</code>	1979
<code>\exp_args:NNx</code>	1141, 2242, 2385	<code>\hskip</code>	130
<code>\exp_args:Nnx</code>	2192	<code>\Hspace</code>	1249
<code>\exp_args:No</code>	3811	<code>\hspace</code>	4092
<code>\exp_args:NV</code>	1720, 2267, 2600, 2627, 2629, 6106	<code>\hss</code>	1803, 2308
<code>\exp_args:Nxx</code>	5596, 5597		
<code>\exp_last_unbraced:NV</code>	1395, 3137, 5889	I	
<code>\exp_not:N</code>	50, 51, 54, 55, 1824, 1868, 1937, 1939, 1944, 1945, 1946, 3034, 3048, 3056, 3058, 3148, 4380, 4391, 4401, 4746, 5182, 5649, 5660, 5738, 5755, 5892, 6285, 6778	<code>\ialign</code>	1102, 1208, 1231, 5814
<code>\exp_not:n</code>	1088, 1707, 3149, 4127, 4128, 4204, 4723, 4734, 4736, 4747, 5612, 5712, 5724, 5725, 5856, 5870, 5882, 5894, 5921, 6287, 6351, 6352	<code>\iddots</code>	1246, 4063, 4064, 4069, 4070
<code>\expandafter</code>	2996	<code>\iddots</code>	79, 1169, 1238
<code>\ExplSyntaxOff</code>	75, 1709, 5330	if commands:	
<code>\ExplSyntaxOn</code>	72, 1702, 5323	<code>\if_mode_math:</code>	269, 4407
<code>\extrarowheight</code>	3597, 5626, 5732, 5748, 6642	<code>\IfBooleanTF</code>	1565
		<code>\ifnum</code>	129, 4372, 5097, 5114
F		<code>\ifstandalone</code>	1535
<code>\fi</code>	129, 1722, 2261, 2264, 4416, 5097, 5114	<code>\ignorespaces</code>	2291, 4418
fi commands:		int commands:	
<code>\fi:</code>	271, 4413	<code>\int_case:nnTF</code>	4037, 4043, 4061, 4067
<code>\five</code>	3207, 3212	<code>\int_compare:nNnTF</code>	143, 144, 168, 902, 903, 914, 921, 957, 967, 1147, 1149, 1287, 1293, 1298, 1577, 1580, 1606, 1610, 1621, 1625, 1626, 1693, 1990, 2190, 2270, 2289, 2475, 2514, 2553, 2625, 2653, 2897, 3274, 3281, 3285, 3287, 3342, 3349, 3353, 3355, 3519, 3558, 3639, 3674, 3676, 4180, 4255, 4359, 4612, 4657, 4675, 4711, 4742, 4759, 4760, 4767, 4768, 4772, 5059, 5069, 5260, 5262, 5264, 5266, 5621, 5623, 5676, 5688, 6015, 6047, 6051, 6124, 6126, 6289, 6291, 6293, 6297, 6299, 6301, 6303, 6305, 6676, 6678
flag commands:		<code>\int_compare_p:n</code>	3421, 3422, 3423, 3424, 4627, 4629, 6116, 6117
<code>\flag_clear_new:n</code>	6771		
<code>\flag_height:n</code>	6796		
<code>\flag_raise:n</code>	6795		
<code>\fontdimen</code>	2442		
fp commands:			
<code>\fp_eval:n</code>	3851		

\int_do_until:nNnn	4567
\int_gadd:Nn	2196, 2288
\int_gincr:N	901,
930, 1538, 1848, 1954, 2047, 2074, 2087,	
2219, 2748, 2773, 2892, 3733, 3781, 5308, 5602	
\int_if_even:nTF	4512, 6796
\int_incr:N	422, 1858, 4595
\int_min:nn	3221,
3223, 3225, 3227, 3237, 3239, 3402, 3429, 3430	
\int_mod:nn	4583
\int_step_inline:nn ...	1450, 1456, 3064,
3070, 3078, 3084, 3219, 4508, 4510, 6689, 6706	
\int_step_inline:nnnn	5192, 5210, 5225, 5228
\c_zero_int	2095, 4425, 4618
iow commands:	
\iow_now:Nn ...	70, 96, 1702, 1703, 1704, 1709
\iow_shipout:Nn	5323, 5324, 5330
\item	2480, 2486
K	
\kern	89
keys commands:	
\keys_define:nn	537,
557, 564, 642, 688, 740, 747, 791, 835,	
849, 866, 879, 1402, 1891, 2180, 4083,	
4321, 4519, 4528, 5297, 5527, 5767, 6141,	
6229, 6269, 6407, 6412, 6493, 6514, 6523, 6887	
\l_keys_key_str	
2181, 6870, 7054, 7060, 7146, 7255, 7266,	
7276, 7291, 7312, 7330, 7347, 7390, 7442, 7493	
\keys_set:nn	
... 567, 569, 583, 824, 827, 834, 1406,	
1414, 1560, 1561, 1925, 2019, 2068, 2189,	
2976, 2988, 3009, 3018, 3158, 3522, 3561,	
3642, 3711, 3759, 4183, 4258, 4275, 4349,	
4523, 4544, 5310, 5850, 6414, 6418, 6546, 6601	
\keys_set_known:nn	
... 4049, 4073, 5578, 6098, 6153, 6174	
\keys_set_known:nnN	2188, 6282
\l_keys_value_tl	7094, 7181, 7188, 7543
L	
\Ldots	1242, 3990, 3993
\ldots	1165, 1234
\leaders	148, 173
\left	1658, 2567, 2582, 6477, 6826, 6850
legacy commands:	
\legacy_if:nTF	672
\line	3131, 7129, 7287
\linewidth	2986
M	
\makebox	2044, 2369
\mathinner	81
mode commands:	
\mode_leave_vertical:	1527, 2340, 4401
msg commands:	
\msg_error:nn	12
\msg_error:nnn	13
\msg_error:nnnn	14, 5832, 5839, 5843
\msg_fatal:nn	15
\msg_fatal:nnn	16
\msg_new:nnn	17
\msg_new:nnnn	18
\msg_redirect_name:nnn	
\multicolumn	1262, 1264, 4094, 4103, 4109, 4131
\multispan	144, 145, 169, 170, 2259
\myfiledate	6
\myfileversion	7
N	
\newcolumntype	3005
\newcounter	382
\NewDocumentCommand	394, 417,
833, 1412, 3157, 3987, 4003, 4019, 4035,	
4059, 4117, 4194, 4272, 4347, 4453, 4462,	
4471, 4480, 4501, 4506, 4521, 4537, 4608,	
4717, 4728, 4740, 6086, 6275, 6323, 6553, 6573	
\NewDocumentEnvironment	1509, 2597,
2606, 2932, 2942, 2972, 2983, 3002, 3014, 5306	
\NewExpandableDocumentCommand	241, 5548
\newlist	398, 409
\NiceArray	2990, 3011, 3020
\NiceArrayWithDelims	2937, 2947
nicematrix commands:	
\g_nicematrix_code_after_tl	285, 681,
2621, 3137, 3140, 5853, 5867, 5879, 6426, 6433	
\g_nicematrix_code_before_tl	1279, 3142,
3149, 4352, 4361, 4721, 4732, 4744, 5890, 5902	
\NiceMatrixLastEnv	241
\NiceMatrixOptions	833, 7348, 7548
\NiceMatrixoptions	7185
\noalign	118, 124, 129, 153, 1129, 1214, 5097, 5274
\nobreak	412
\normalbaselines	1194
\NotEmpty	1258
\null	2287
\nulldelimiterspace	2578, 2590, 6462, 6668
\nullfont	6473, 6480, 6822, 6829, 6846, 6853
\numexpr	181, 182
O	
\omit	143, 2655, 2671, 2747, 2772, 6422, 6423
\OnlyMainNiceMatrix	1254, 4751
P	
\par	2474, 2482
\path	6241
peek commands:	
\peek_meaning:NNTF	186, 424, 1159, 2998
\peek_meaning_ignore_spaces:NNTF	2599, 5100
\peek_meaning_remove_ignore_spaces:NNTF	176
\peek_remove_spaces:n	
... 4719, 4730, 5550, 6555, 6575	
\pgfdeclareshape	3205
\pgfextracty	6018
\pgfgetlastxy	486
\pgfpathcircle	3975
\pgfpathlineto	4884, 4890,
5037, 5043, 6213, 6226, 6377, 6683, 6700, 6734	
\pgfpathmoveto	4883, 4889,
5036, 5042, 6212, 6225, 6372, 6682, 6699, 6725	
\pgfpathrectanglecorners	4685, 4877, 5030, 6132
\pgfpointhead	484
\pgfpointanchor	191,</

skip commands:

<code>\skip_gadd:Nn</code>	2719
<code>\skip_gset:Nn</code>	2710
<code>\skip_gset_eq:NN</code>	2717, 2718
<code>\skip_horizontal:N</code>	149, 174, 1331, 1332, 1572, 1573, 1628, 1629, 1665, 1666, 1669, 1670, 1688, 1689, 1729, 1735, 1824, 2226, 2239, 2246, 2382, 2389, 2591, 2592, 2594, 2595, 2666, 2667, 2679, 2681, 2697, 2699, 2721, 2728, 2730, 2749, 2754, 2756, 2774, 2783, 2788, 2790, 2792, 2794, 2820, 2878, 2879, 2880, 2883, 2918, 2923, 2924, 2925
<code>\skip_horizontal:n</code>	444, 1077, 1870
<code>\skip_vertical:N</code>	153, 1111, 1113, 1661, 1672, 2471, 2496, 5274
<code>\skip_vertical:n</code>	972, 5107
<code>\g_tmpa_skip</code>	2710, 2717, 2718, 2719, 2721, 2749, 2774
<code>\c_zero_skip</code>	1219
<code>\smash</code>	6859, 6860
<code>\space</code>	282, 283
<code>\stepcounter</code>	430, 435
str commands:	
<code>\c_backslash_str</code>	282
<code>\c_colon_str</code>	832
<code>\str_case:nn</code>	1942, 5637, 6025, 6037, 6062, 6074, 6719, 6728
<code>\str_case:nnTF</code>	1633, 1788, 2295, 2428, 5165, 6793, 6806
<code>\str_clear_new:N</code>	6600
<code>\str_gclear:N</code>	3153
<code>\str_gset:Nn</code>	1524, 2936, 2945, 2974, 2987, 3006, 3016, 6314
<code>\str_if_empty:NTF</code>	939, 1042, 1120, 1523, 2663, 2688, 2706, 2737, 2763, 2809, 2830, 2935, 2944, 3076, 3233, 3245, 5493, 5520, 5573, 5940, 5945, 6740, 6754, 6764
<code>\str_if_eq:nnTF</code>	104, 281, 1100, 1816, 1819, 1863, 1886, 1916, 1933, 1936, 1949, 1950, 1951, 2009, 2052, 2097, 2129, 2151, 2174, 2233, 2376, 2611, 2613, 2646, 4604, 6104, 6431
<code>\str_if_eq_p:nn</code>	543, 1752, 1753, 1829, 4651, 4655, 4705, 4709, 5563, 5568
<code>\str_if_in:NnTF</code>	2416, 2528
<code>\str_new:N</code>	252, 277, 331, 524, 831
<code>\str_range:Nnn</code>	2420, 2532
<code>\str_set:Nn</code>	253, 332, 674, 820, 1844, 1893, 1895, 1897, 1899, 1901, 1904, 1906, 1911, 1924, 1937, 1939, 2017, 2018, 2035, 2184, 2317, 2360, 5529, 5531, 5533, 5535, 5537, 5539, 5541, 5543, 5574, 5577, 5628, 5735, 5751, 5788, 5790, 5792, 5794, 5797, 5800, 5803, 5805, 6048, 6052, 6540
<code>\str_set_eq:NN</code>	678, 832, 5575
<code>\l_tmpa_str</code>	674, 675, 677, 678
<code>\strut</code>	2480, 2486
<code>\strutbox</code>	3597, 3615, 6642, 6645
<code>\SubMatrix</code>	1387, 3118, 6566, 7070, 7095, 7102, 7108, 7112, 7135, 7294

sys commands:

<code>\sys_if_engine_xetex_p:</code>	1074
<code>\sys_if_output_dvi_p:</code>	1074

T

<code>\tabcolsep</code>	1094, 1665, 1669
<code>\tabskip</code>	1219
<code>\tabularnote</code>	394, 417, 424, 7213, 7233
<code>\tabularnotes</code>	2485
T _E X and L ^A T _E X 2 _ε commands:	
<code>\@BTnormal</code>	1192
<code>\@addamp</code>	1495, 2261
<code>\@addamp@LaTeX</code>	1495
<code>\@addtopreamble</code>	2268
<code>\@array</code>	1502, 1506
<code>\@array@array</code>	1502
<code>\@arraycr</code>	1499
<code>\@arraycr@array</code>	1499
<code>\@arstrut</code>	2285
<code>\@arstrutbox</code>	932, 933, 972, 1221, 1223, 1225, 1228, 1230, 1967, 1978, 1993, 1999, 2342, 2346
<code>\@classx</code>	1497
<code>\@classx@array</code>	1497
<code>\@currenvir</code>	6431
<code>\@depth</code>	6483, 6831, 6855
<code>\@empty</code>	2268
<code>\@finalstrut</code>	1978
<code>\@firstampfalse</code>	2261
<code>\@gobblethree</code>	74
<code>\@halignto</code>	1097, 1098
<code>\@height</code>	132, 148, 173, 6481, 6830, 6854
<code>\@ifclassloaded</code>	59, 62, 6925, 6935
<code>\@ifnextchar</code>	1270, 1506
<code>\@ifpackageloaded</code>	32, 35, 38, 41, 44, 47, 95, 111, 195, 6928, 6938
<code>\@mainaux</code>	70, 96, 1702, 1703, 1704, 1709, 5323, 5324, 5330
<code>\@mkpream</code>	1504, 2267
<code>\@mkpream@array</code>	1504
<code>\@preamble</code>	2286
<code>\@preamerr</code>	2261
<code>\@sharp</code>	2284
<code>\@tabarray</code>	1099, 1506
<code>\@tabular</code>	1503
<code>\@tabular@array</code>	1503
<code>\@tempswafalse</code>	1722, 2264
<code>\@tempswattrue</code>	1721, 2263
<code>\@temptokena</code>	208, 209, 225, 228, 1720, 1740, 2262, 2266
<code>\@whiles</code>	1722, 2264
<code>\@width</code>	132, 6484, 6832, 6856
<code>\@xargarraycr</code>	1500
<code>\@xargarraycr@array</code>	1500
<code>\@xarraycr</code>	1498
<code>\@xarraycr@array</code>	1498
<code>\@xhline</code>	135
<code>\array@array</code>	1501
<code>\bBigg@</code>	1325, 1327
<code>\c@MaxMatrixCols</code>	2964, 6994
<code>\c@tabularnote</code>	2464, 2475, 2503
<code>\col@sep</code>	1094, 1095, 1628, 1689, 2666, 2719, 2783, 2883, 2918, 3484, 3506
<code>\CT@arc</code>	115, 116
<code>\CT@arc@</code>	114, 119, 133, 147, 172, 188, 190, 1531, 2497, 3155, 4892, 5045, 6105, 6192, 6379, 6673

<code>\CT@drs</code>	121, 122	<code>\tl_gset:Nn</code>	228, 232, 234, 1516, 1517, 1518, 1706, 1728, 1734, 2099, 2100, 2130, 2156, 3148, 4428
<code>\CT@drsc@</code>	125, 4867, 4868, 4872, 5022, 5023, 5027	<code>\tl_if_blank:nTF</code>	4455, 4458, 4464, 4467, 4473, 4476, 4482, 4485, 4592, 4785, 4819, 4940, 4974, 5552
<code>\CT@everycr</code>	1212	<code>\tl_if_blank_p:n</code>	4650, 4654, 4704, 4708, 4868, 5023, 5562, 5567
<code>\CT@row@color</code>	1214	<code>\tl_if_empty:NTF</code>	1137, 1554, 1563, 1656, 1674, 2474, 3116, 3117, 3142, 4350, 4398, 4589, 4672, 4673, 4798, 4802, 4813, 4953, 4957, 4968, 5620, 5865, 5887, 6102, 6475, 6671, 6824, 6848
<code>\endarray@array</code>	1505	<code>\tl_if_empty:nTF</code>	160, 655, 795, 837, 853, 869, 890, 1475, 2608, 2635, 3523, 3562, 3643, 3712, 3760, 4184, 4259, 4276, 6532, 6791, 6859, 6974
<code>\if@firstamp</code>	2261	<code>\tl_if_empty_p:N</code>	1763, 1775, 3839, 3840
<code>\if@tempswa</code>	1722, 2264	<code>\tl_if_empty_p:n</code>	2465, 5593
<code>\insert@column</code>	1496	<code>\tl_if_eq:NNTF</code>	3804
<code>\insert@column@array</code>	1496	<code>\tl_if_eq:NnTF</code> ..	1139, 1726, 2236, 2379, 2404, 2523, 4923, 5064, 5074, 5092, 6688, 6705
<code>\NC@</code>	1158, 2997	<code>\tl_if_eq:nnTF</code> ...	667, 811, 2127, 2149, 4424
<code>\NC@do</code>	2996	<code>\tl_if_exist:NTF</code>	1548
<code>\NC@find</code>	210, 226	<code>\tl_if_in:NnTF</code>	4558, 4646, 4669, 4700
<code>\NC@find@V</code>	1719	<code>\tl_if_in:nnTF</code>	370, 2114, 2124, 2139
<code>\NC@list</code>	1722, 2264, 2996	<code>\tl_if_single_token:nTF</code>	606, 819
<code>\NC@rewrite@S</code>	206, 227	<code>\tl_if_single_token_p:n</code>	66
<code>\new@ifnextchar</code>	1270	<code>\tl_item:Nn</code>	231, 232, 234
<code>\newcol@</code>	1160, 1161, 2999, 3000	<code>\tl_map_inline:nn</code>	2609
<code>\nicematrix@redefine@check@rerun</code> ..	96, 99	<code>\tl_new:N</code>	230, 233, 266, 273, 285, 286, 287, 290, 297, 299, 324, 325, 329, 334, 384, 509, 513, 532, 534, 535
<code>\pgf@relevantforpicturesizefalse</code>	1353, 1467, 3192, 3803, 3972, 4437, 4554, 4857, 4900, 5012, 5053, 5413, 5423, 5434, 5928, 6101, 6191, 6366, 6441, 6604	<code>\tl_put_left:Nn</code>	1192, 1400
<code>\pgfsys@getposition</code>	1351, 1356, 1362, 1429, 1437, 1440	<code>\tl_put_right:Nn</code> ..	657, 1202, 1300, 1340, 1557
<code>\pgfsys@markposition</code>	1059, 1067, 1112, 1204, 1350, 2659, 2680, 2698, 2729, 2755, 2791, 2823	<code>\tl_range:nnn</code>	104
<code>\pgfutil@check@rerun</code>	101, 102	<code>\tl_set:Nn</code> ..	231, 4497, 4498, 4560, 4576, 4579, 4656, 4658, 4674, 4676, 4710, 4712, 4781, 5579, 6125, 6127, 6158, 6159, 6187, 6188
<code>\reserved@a</code>	134	<code>\tl_set_eq:NN</code>	379, 511, 4494, 4495, 4659, 4799, 4954, 6155, 6156, 6184, 6185, 6560, 6561, 6590, 6591, 6593, 6594
<code>\rvtx@iffformat@geq</code>	65	<code>\tl_set_rescan:Nnn</code>	2623, 3986, 4116, 4193, 4271, 6572
<code>\set@color</code>	5621, 5924	<code>\tl_to_str:n</code>	6958
<code>\tikz@library@external@loaded</code>	1532	<code>\g_tmpa_tl</code>	228, 231, 234
tex commands:		tmpc commands:	
<code>\tex_mkern:D</code>	83, 85, 87, 90	<code>\l_tmpc_dim</code> ..	304, 1482, 1489, 3226, 3230, 4662, 4663, 4686, 4863, 4874, 4879, 4884, 4890, 5018, 5032, 5037, 5043, 5934, 5939, 5994, 6122, 6133, 6210, 6213, 6374, 6377, 6385
<code>\tex_the:D</code>	209	<code>\l_tmpc_int</code>	4565, 4566, 4567
<code>\textfont</code>	2442	<code>\l_tmpc_tl</code>	4342, 4350, 4355, 4364, 4492, 4494, 4497, 4659, 4678, 4782, 4798, 4799, 4802, 4807, 4809, 4813, 4818, 4820, 4937, 4953, 4954, 4957, 4962, 4964, 4968, 4973, 4975, 6155, 6163, 6165, 6184, 6201, 6207, 6558, 6560, 6564
<code>\textit</code>	386	tmpd commands:	
<code>\textsuperscript</code>	387, 388	<code>\l_tmpd_dim</code>	305, 1484, 1490, 3228, 3231, 4683, 4686, 4875, 4879, 5028, 5032, 5936, 5939, 5972, 5983, 5987, 5990, 5994, 6131, 6134, 6376, 6377, 6395
<code>\the</code>	181, 182, 1722, 1740, 2264, 2266, 2996		
<code>\thetabularnote</code>	389		
<code>\tikzexternaldisable</code>	1534		
<code>\tikzset</code>	1372, 1536, 3121		
tl commands:			
<code>\c_empty_tl</code>	4344, 5809		
<code>\tl_clear:N</code>	4809, 4820, 4964, 4975, 6096		
<code>\tl_clear_new:N</code> ..	4492, 4493, 4542, 4578, 4782, 4937, 6558, 6559, 6585, 6586, 6587, 6588		
<code>\tl_const:Nn</code>	50, 51, 54, 55, 510, 2841, 2886, 6782		
<code>\tl_count:n</code>	2423, 2535		
<code>\tl_gclear:N</code>	899, 1546, 1553, 1724, 2265, 3133, 3140, 4442		
<code>\tl_gclear_new:N</code>	1273, 1274, 1275, 1276, 1277, 1278, 1279, 1545		
<code>\tl_gput_left:Nn</code>	1082, 1757, 4744		
<code>\tl_gput_right:Nn</code>	1082, 1582, 1769, 1823, 1866, 1880, 2112, 2133, 2141, 2154, 2160, 2166, 2227, 3032, 3046, 3054, 3146, 4119, 4196, 4352, 4361, 4373, 4378, 4389, 4400, 4430, 4721, 4732, 5112, 5180, 5279, 5605, 5853, 5867, 5879, 5890, 5902, 5914, 6344		

<code>\l_tmpd_tl</code>	4493, 4495, 4498, 6156, 6160, 6168, 6185, 6197, 6218, 6559, 6561, 6564	<code>\vbox_set_top:Nn</code>	969
token commands:		<code>\vbox_to_ht:nn</code>	464, 491, 2570, 2583
<code>\token_to_str:N</code>	6946, 6947, 6975, 7065, 7070, 7076, 7094, 7102, 7108, 7112, 7129, 7135, 7153, 7166, 7172, 7213, 7220, 7233, 7250, 7266, 7278, 7287, 7293, 7331, 7348, 7548	<code>\vbox_to_zero:n</code>	971
U		<code>\vcenter</code>	1659, 2568, 6478, 6827, 6851, 7250
<code>\unskip</code>	2490	<code>\Vdots</code>	1244, 4022, 4025
use commands:		<code>\vdots</code>	1167, 1236
<code>\use:N</code>	1085, 1335, 1336, 1551, 1569, 1570, 2959, 2979, 4441	<code>\Vdotsfor</code>	1251
<code>\use:n</code>	1930, 4277, 4751, 5647, 5658, 5736, 5753, 6162, 6167, 6283, 6777	<code>\vfill</code>	467, 493
<code>\usepackage</code>	6930, 6940	<code>\vline</code>	133
<code>\usepgfmodule</code>	2	<code>\VNiceMatrix</code>	6255
V		<code>\vNiceMatrix</code>	6252
vbox commands:		<code>\vrule</code>	131, 1967, 2342, 2346
<code>\vbox:n</code>	89	<code>\vskip</code>	130
		<code>\vtop</code>	1109
		X	
		<code>\xglobal</code>	917, 924, 1187, 2860, 2904
		Z	
		<code>\Z</code>	6535

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	6
4.3	The mono-row blocks	6
4.4	The mono-cell blocks	6
4.5	Horizontal position of the content of the block	7
5	The rules	7
5.1	Some differences with the classical environments	8
5.1.1	The vertical rules	8
5.1.2	The command <code>\cline</code>	8
5.2	The thickness and the color of the rules	9
5.3	The tools of nicematrix for the rules	9
5.3.1	The keys <code>hlines</code> and <code>vlines</code>	9
5.3.2	The keys <code>hvlines</code> and <code>hvlines-except-borders</code>	10
5.3.3	The (empty) corners	10
5.4	The command <code>\diagbox</code>	11
5.5	Dotted rules	11
6	The color of the rows and columns	12
6.1	Use of <code>colortbl</code>	12
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	12
6.3	Color tools with the syntax of <code>colortbl</code>	16
7	The command <code>\RowStyle</code>	17

8	The width of the columns	17
8.1	Basic tools	17
8.2	The columns V of varwidth	19
8.3	The columns X	19
9	The exterior rows and columns	20
10	The continuous dotted lines	21
10.1	The option nullify-dots	23
10.2	The commands \Hdotsfor and \Vdotsfor	23
10.3	How to generate the continuous dotted lines transparently	24
10.4	The labels of the dotted lines	25
10.5	Customisation of the dotted lines	25
10.6	The dotted lines and the rules	26
11	The \CodeAfter	27
11.1	The command \line in the \CodeAfter	27
11.2	The command \SubMatrix in the \CodeAfter	27
12	The notes in the tabulars	30
12.1	The footnotes	30
12.2	The notes of tabular	30
12.3	Customisation of the tabular notes	32
12.4	Use of {NiceTabular} with threeparttable	33
13	Other features	34
13.1	Use of the column type S of siunitx	34
13.2	Alignment option in {NiceMatrix}	34
13.3	The command \rotate	34
13.4	The option small	35
13.5	The counters iRow and jCol	35
13.6	The option light-syntax	36
13.7	Color of the delimiters	36
13.8	The environment {NiceArrayWithDelims}	36
14	Use of Tikz with nicematrix	37
14.1	The nodes corresponding to the contents of the cells	37
14.1.1	Les colonnes V de varwidth	38
14.2	The “medium nodes” and the “large nodes”	38
14.3	The nodes which indicate the position of the rules	40
14.4	The nodes corresponding to the command \SubMatrix	41
15	API for the developers	41
16	Technical remarks	42
16.1	Definition of new column types	42
16.2	Diagonal lines	42
16.3	The “empty” cells	43
16.4	The option exterior-arraycolsep	43
16.5	Incompatibilities	44
17	Examples	44
17.1	Utilisation of the key “tikz” of the command \Block	44
17.2	Notes in the tabulars	45
17.3	Dotted lines	46
17.4	Dotted lines which are no longer dotted	47
17.5	Stacks of matrices	48
17.6	How to highlight cells of a matrix	51
17.7	Utilisation of \SubMatrix in the \CodeBefore	53

18	Implementation	54
19	History	224
	Index	232