

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

July 16, 2022

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.

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

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, TeX Live or MacTeX.

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

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

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

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

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

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

*This document corresponds to the version 6.11 of `nicematrix`, at the date of 2022/07/16.

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

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. 21

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.

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

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

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

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

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

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

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

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

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 keys `hlines`, `vlines` and `hvlines` draw all the corresponding rules in the block;⁸
- the key `line-width` is the width of the rules (this key is meaningful only when one of the keys `draw`, `hvlines`, `vlines` and `hlines` is used);
- 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 pt⁹);
- 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 `\\`);
- when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz¹⁰ 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. 47;
- the key `name` provides a name to the rectangular Tikz node corresponding to the block; it's possible to use that name with Tikz in the `\CodeAfter` of the environment (cf. p. 28);
- the key `respect-arraystretch` prevents the setting of `\arraystretch` to 1 at the beginning of the block (which is the behaviour by default) ;
- 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`; it's possible, in fact, in the list which is the value of the key `borders`, to add an entry of the form `tikz={list}` where `list` is a list of couples `key=value` of Tikz specifying the graphical characteristics of the lines that will be drawn (for an example, see p. 51).

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}
      & {\LARGE Some beautiful flowers}
      & & marigold \\
iris & & & lis \\
arum & periwinkle & forget-me-not & hyacinth
\end{NiceTabular}
```

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

⁸However, the rules are not drawn in the sub-blocks of the block, as always with `nicematrix`: the rules are not drawn in the blocks (cf. section 5 p. 7).

⁹This value is the initial value of the *rounded corners* of Tikz.

¹⁰Tikz 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.
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).

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

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`.¹¹
- It's possible to draw one or several borders of the cell with the key `borders`.

¹¹If 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.¹²

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 positioning 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.

¹²One 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}}
```

5.1.2 The command `\cline`

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

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

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

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

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

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

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

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

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`¹³ presented p. 4;
 - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `\Vdots`, etc. (cf. p. 23).
- 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.¹⁴

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

¹³And also the command `\multicolumn` but it's recommended to use instead `\Block` in the environments of `nicematrix`.

¹⁴It'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.¹⁵

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

A 6x6 grid with blue shaded cells. The unshaded cells are at (2,3), (2,4), (2,5), (3,3), (3,4), (3,5), (4,3), (4,4), (4,5), (5,3), (5,4), (5,5), (6,3), (6,4), (6,5). The letters 'A' and 'B' are placed in the unshaded cells. 'A' is at (2,3), (2,4), (2,5), (3,3), (3,4), (3,5), (4,3), (4,4), (4,5), (5,3), (5,4), (5,5), (6,3), (6,4), (6,5). 'B' is at (5,3).

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).

```
\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		

¹⁵For 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. The precise definition of a “non-empty cell” is given below (cf. p. 46).

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. 14).

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.

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

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

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

5.5 Commands for customized rules

It's also possible to define commands and letters for customized rules with the key `custom-line` available in `\NiceMatrixOptions` and in the options of individual environments. That key takes in as argument a list of `key=value` pairs. First, there is two keys to define the tools which will be used to use that new type of rule.

- the key `command` is the name (without the backslash) of a command that will be created by `nicematrix` and that will be available for the final user in order to draw horizontal rules (similarly to `\hline`);
- **New 6.11** the key `ccommand` is the name (without the backslash) of a command that will be created by `nicematrix` and that will be available for the final user to order to draw partial horizontal rules (similarly to `\cline`, hence the name `ccommand`): the argument of that command is a list of intervals of columns specified by the syntax *i* or *i-j*.¹⁶
- the key `letter` takes in as argument a letter¹⁷ that the user will use in the preamble of an environment with preamble (such as `\NiceTabular`) in order to specify a vertical rule.

For the description of the rule itself, there is three possibilities.

- *First possibility*

It's possible to specify composite rules, with a color and a color for the inter-rule space (as possible with `colortbl` for instance).

¹⁶It's recommended to use such commands only once in a row because each use will create space between the rows corresponding to the total width of the rule.

¹⁷The following letters are forbidden: `lcrpmbVX|()[]!@<>`

- the key `multiplicity` is the number to consecutive rules that will be drawn: for instance, a value of 2 will create double rules such those created by `\hline\hline` or `||` in the preamble of an environment;
- the key `color` sets the color of the rule ;
- the key `sep-color` sets the color between two successive rules (should be used only in conjunction with `multiplicity`).

That system may be used, in particular, for the definition of commands and letters to draw rules with a specific color (and those rules will respect the blocks and corners as do all the rules of `nicematrix`).

```
\begin{NiceTabular}{lcIcIc}[custom-line = {letter=I, color=blue}]
\hline
      & \Block{1-3}{dimensions} \\
      & L & l & h \\
\hline
Product A & 3 & 1 & 2 \\
Product B & 1 & 3 & 4 \\
Product C & 5 & 4 & 1 \\
\hline
\end{NiceTabular}
```

- *Second possibility*

It's possible to use the key `tikz` (if Tikz is loaded). In that case, the rule is drawn directly with Tikz by using as parameters the value of the key `tikz` which must be a list of *key=value* pairs which may be applied to a Tikz path.

By default, no space is reserved for the rule that will be drawn with Tikz. It is possible to specify a reservation (horizontal for a vertical rule and vertical for an horizontal one) with the key `total-width`. That value of that key, is, in some ways, the width of the rule that will be drawn (`nicematrix` does not compute that width from the characteristics of the rule specified in `tikz`).

	dimensions		
	L	l	H
Product A	3	1	2
Product B	1	3	4
Product C	5	4	1

Here is an example with the key `dotted` of Tikz.

```
\NiceMatrixOptions
{
  custom-line =
  {
    letter = I ,
    tikz = dotted ,
    total-width = \pgflinewidth
  }
}

\begin{NiceTabular}{cIcIc}
one & two & three \\
four & five & six \\
seven & eight & nine
\end{NiceTabular}
```

one	two	three
four	five	six
seven	eight	nine

- *Third possibility* : the key `dotted`

As one can see, the dots of a dotted line of Tikz have the shape of a square, and not a circle. That's why the extension `nicematrix` provides in the key `custom-line` a key `dotted` which will draw rounded dots. The initial value of the key `total-width` is, in this case, equal to the diameter of the dots (but the user may change the value with the key `total-width` if needed). Those dotted rules are also used by `nicematrix` to draw continuous dotted rules between cells of the matrix with `\Cdots`, `\Vdots`, etc. (cf. p. 23).

In fact, `nicematrix` defines by default the commands `\hdottedline` and `\cdottedline` and the letter “:” for those dotted rules.¹⁸

```
\NiceMatrixOptions % present in nicematrix.sty
{
  custom-line =
  {
    letter = : ,
    command = hdottedline ,
    ccommand = cdottedline ,
    dotted
  }
}
```

Thus, it's possible to use the commands `\hdottedline` and `\cdottedline` to draw horizontal dotted rules.

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

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

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

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

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

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`).

¹⁸However, it's possible to overwrite those definitions with a `custom-line` (in order, for example, to switch to dashed lines).

- 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.¹⁹

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}
```

New commands are available in that `\CodeBefore`: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors`, `\rowlistcolors`, `\chessboardcolors` and `arraycolor`.²⁰

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

¹⁹If you use Overleaf, Overleaf will do automatically the right number of compilations.

²⁰Remark 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. 43.

- 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. 21). It's only a particular case of `\rectanglecolor`.
- The command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

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

1	-1	1
-1	1	-1
1	-1	1

We have used the key `r` which aligns all the columns rightwards (cf. p. 37).

- 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).

```
$$\begin{NiceArray}{l1l1}[hvlines]
\CodeBefore
  \rowcolor{red!15}{1,3-5,8-}
\Body
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3 \\
a_4 & b_4 & c_4 \\
a_5 & b_5 & c_5 \\
a_6 & b_6 & c_6 \\
a_7 & b_7 & c_7 \\
a_8 & b_8 & c_8 \\
a_9 & b_9 & c_9 \\
a_{10} & b_{10} & c_{10}
\end{NiceArray}$$
```

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

- 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`²¹. 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.²²
- With the key `respect-blocks` the “rows” alternately colored may extend over several rows if they have to incorporate blocks (created with the command `\Block`: cf. p. 4).

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

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

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

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

- 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.

²¹The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

²²Otherwise, the color of a given row relies only upon the parity of its absolute number.


```

\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]{lSSSS}
\CodeBefore
  \rowcolor{red!15}{1-2}
  \rowcolors{3}{blue!15}{}
\Body
\toprule
\Block{2-1}{Product} &
\Block{1-3}{dimensions (cm)} & & &
\Block{2-1}{\rotate Price} \\
\cmidrule{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.²³

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

- `\cellcolor` which colorizes a cell;²⁴
- `\rowcolor` which must be used in a cell and which colorizes the end of the row;
- `\columncolor` which must be used in the preamble of the environment with the same syntax as the corresponding command of `colortbl` (however, unlike the command `\columncolor` of `colortbl`, this command `\columncolor` can appear within another command, itself used in the preamble of the array).

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

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

7 The 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.

- The key `nb-rows` sets the number of rows to which the specifications of the current command will apply (with the special value `*`, it will apply to all the following rows).
- 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).
- The key `rowcolor` sets the color of the background and the key `color` sets the color of the text.²⁵

²³Up to now, this key is *not* available in `\NiceMatrixOptions`.

²⁴However, this command `\cellcolor` will delete the following spaces, which does not the command `\cellcolor` of `colortbl`.

²⁵The 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).

- The key `bold` enforces bold characters for the cells of the row, both in math 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
1	2	3	4
I	II	III	IV

The command `\rotate` is described p. 37.

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`.

```
\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. 21) 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.²⁶

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

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

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}
$\begin{pNiceMatrix}
a & b \\ c & d
\end{pNiceMatrix}
=
\begin{pNiceMatrix}
1 & 1245 \\ 345 & 2
\end{pNiceMatrix}$
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 1245 \\ 345 & 2 \end{pmatrix}$$

²⁶The 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).

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`²⁷. 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}
```

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

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

8.2 The columns V of varwidth

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. 41. If the content of the cell is empty, the cell will be considered as empty by `nicematrix` in the construction of the dotted lines and the «empty corners» (that's not the case with a cell of a column `p`, `m` or `b`).

```
\begin{NiceTabular}[corners=NW,hvlines]{V{3cm}V{3cm}V{3cm}}
& some very very very long text & some very very very long text \\
some very very very long text & \\
some very very very long text & \\
\end{NiceTabular}
```

²⁷At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

	some very very very long text	some very very very long text
some very very very long text		
some very very very long text		

One should remark that the extension `varwidth` (at least in its version 0.92) has some problems: for instance, with LuaLaTeX, it does not work when the content begins with `\color`.

8.3 The columns X

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.²⁸

As with the packages `tabu`²⁹ 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).³⁰
- 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`.

```
\begin{NiceTabular}[width=9cm]{X[2,l]X[l]}[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”.

²⁸If `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`).

²⁹The extension `tabu` is now considered as deprecated.

³⁰The 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.

```

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

```

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

The dotted lines have been drawn with the tools presented p. 23.

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.³¹
- 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. 39) 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 and the number of columns.
 - In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

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

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

```

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

```

³¹The users wishing exterior columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 29).

```

& a_{31} & a_{32} & a_{33} & a_{34} & \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
& C_1 & \Cdots & & C_4 & \\
\end{pNiceArray}$

```

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

Remarks

- As shown in the previous example, the horizontal and vertical rules don't extend in the exterior rows and columns. This remark also applies to the customized rules created by the key `custom-line` (cf. p. 11).
- 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. 19) 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. 29.

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`.³²

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells³³ 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`.³⁴

```

\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}

```

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

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

```

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

```

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

³²The 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`.

³³The precise definition of a “non-empty cell” is given below (cf. p. 46).

³⁴It'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. 27.

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

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

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

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

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

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\Vdots` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.³⁵

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

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

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

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.

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

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

³⁵In `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. 19

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.

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

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

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

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

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`³⁶ 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}
```

³⁶We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

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

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`.³⁷

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`³² and `\hdotsfor` are redefined within the environments provided by `nicematrix` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` (“automatic dots” of `amsmath`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

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

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

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

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

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. 29) 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^{n \text{ times}} & & & \\
0 & & & & 1 & \\
\end{bNiceMatrix}
```

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

³⁷The 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`.

10.5 Customisation of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and `\Vdotsfor` (and by the command `\line` in the `\CodeAfter` which is described p. 29) may be customized by the following options (specified between square brackets after the command):

- `color`;
- `radius`;
- `shorten-start`, `shorten-end` and `shorten`;
- `inter`;
- `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` (*xdots* to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.), and, thus have for names:

- `xdots/color`;
- `xdots/radius`;
- `xdots/shorten-start`, `xdots/shorten-end` and `xdots/shorten`;
- `xdots/inter`;
- `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. 21.

New 6.9 The option `xdots/radius`

The option `radius` fixes the radius of the dots. The initial value is 0.53 pt.

The option `xdots/shorten`

The keys `xdots/shorten-start` and `xdots/shorten-end` fix the margin at the extremities of the line. The key `xdots/shorten` fixes both parameters. The initial value is 0.3 em (it is recommended to use a unit of length dependent of the current font).

New 6.10 The keys `xdots/shorten-start` and `xdots/shorten-end` have been introduced in version 6.10. In the previous versions, there was only `xdots/shorten`.

New 6.9 The option `xdots/inter`

The option `xdots/inter` fixes the length between the dots. The initial value is 0.45 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).³⁸

```
\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

³⁸The 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. Nevertheless, you can have a look at the following page to see how to have dotted rules with rounded dots in Tikz: <https://tex.stackexchange.com/questions/52848/tikz-line-with-large-dots>

(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 paths (with the exception of “color”, “shorten >” and “shorten <”).

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

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

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

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`, by the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` and by the tools created by `custom-line` are not drawn within the blocks).³⁹

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

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

11 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.⁴⁰

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 in that optional argument 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. 40.

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

³⁹On the other side, the command `\line` in the `\CodeAfter` (cf. p. 29) does *not* create block.

⁴⁰There is also a key `code-before` described p. 14.

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

The command `\line` draws directly dotted lines between cells or blocks. It takes in two arguments for the cells or blocks to link. Both argument may be:

- a specification of cell of the form i - j where i is the number of the row and j is the number of the column;
- **New 6.10** the name of a block (created by the command `\Block` with the key `name` of that command).

The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 27).

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}
```

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

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. 45).

```
\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}
```

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

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 `.`;
- 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.⁴¹

⁴¹There 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 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.

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

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

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

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

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

```

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

```

$$\begin{pmatrix} \frac{1}{2} \\ \frac{1}{4} \end{pmatrix}$$

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

There is also a key `name` which gives a name to the submatrix created by `\SubMatrix`. That name is used to create PGF/Tikz nodes: cf p. 44.

It's also possible to specify some delimiters⁴² by placing them in the preamble of the environment (for the environments with a preamble: `{NiceArray}`, `{pNiceArray}`, etc.). This syntax is inspired by the extension `blkarray`.

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

```

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

```

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

11.3 The commands `\OverBrace` and `\UnderBrace` in the `\CodeAfter`

The commands `\OverBrace` and `\UnderBrace` provide a way to put horizontal braces on a part of the array. These commands take in three arguments:

- the first 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 second argument is the lower-right corner with the same syntax;
- the third argument is the label of the brace that will be put by `nicematrix` (with PGF) above the brace (for the command `\OverBrace`) or under the brace (for `\UnderBrace`).

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
11 & 12 & 13 & 14 & 15 & 16 \\
\CodeAfter
  \OverBrace{1-1}{2-3}{A}
  \OverBrace{1-4}{2-6}{B}
\end{pNiceMatrix}

```

$$\begin{pmatrix} \overbrace{1 \quad 2 \quad 3}^A & \overbrace{4 \quad 5 \quad 6}^B \\ 11 & 12 & 13 & 14 & 15 & 16 \end{pmatrix}$$

In fact, the commands `\OverBrace` and `\UnderBrace` take in an optional argument (in first position and between square brackets) for a list of `key=value` pairs. The available keys are:

⁴²Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

- `left-shorten` and `right-shorten` which do not take in value; when the key `left-shorten` is used, the abscissa of the left extremity of the brace is computed with the contents of the cells of the involved sub-array, otherwise, the position of the potential vertical rule is used (idem for `right-shorten`).
- `shorten`, which is the conjunction of the keys `left-shorten` and `right-shorten`;
- `yshift`, which shifts vertically the brace (and its label) ;
- `color`, which sets the color of the brace (and its label).

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 & 6 & \\
11 & 12 & 13 & 14 & 15 & 16 & \\
\CodeAfter
  \OverBrace[shorten,yshift=3pt]{1-1}{2-3}{A}
  \OverBrace[shorten,yshift=3pt]{1-4}{2-6}{B}
\end{pNiceMatrix}

```

$$\begin{array}{ccccc}
 & \overbrace{\hspace{1.5cm}}^A & & \overbrace{\hspace{1.5cm}}^B & \\
 \left(\begin{array}{cccccc}
 1 & 2 & 3 & 4 & 5 & 6 \\
 11 & 12 & 13 & 14 & 15 & 16
 \end{array} \right)
 \end{array}$$

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 specified by `first-col` and `last-col`). 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.
- New 6.8** If several commands `\tabularnote` are used in a tabular with the same argument, only one note is inserted at the end of the tabular (but all the labels are composed, of course). It's possible to control that feature with the key `notes/detect-duplicates`.
- 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. 34. 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\tabularnote{This note is shared by two references.} & 90 \\
Schoelcher & Victor & 89\tabularnote{The label of the note is overlapping.}\\
\end{NiceTabular}
\end{table}
```

```

Touchet & Marie\tabularnote{This note is shared by two references.} & 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 ^d	90
Schoelcher	Victor	89 ^e
Touchet	Marie ^d	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 This note is shared by two references.

^e 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`. This style of list is defined as follows (with, of course, keys of `enumitem`):

```
noitemsep , leftmargin = * , align = left , labelsep = 0pt
```

The specification `align = left` in that style 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. 34).

The key `notes/enumitem-keys` specifies a list of pairs `key=value` (following the specifications of `enumitem`) to customize that style of list (it uses internally the command `\setlist*` of `enumitem`).

- 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`.

Initially, the style of list is defined by: `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`).

- The key `notes/detect-duplicates` activates the detection of the commands `\tabularnotes` with the same argument.

Initial value : `true`

For an example of customisation of the tabular notes, see p. 48.

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

14 Autres fonctionnalités

14.1 Command `\ShowCellNames`

New 6.9 The command `\ShowCellNames`, which may be used in the `\CodeBefore` and in the `\CodeAfter` display the name (with the form *i-j*) of each cell.

```
\begin{NiceTabular}{ccc}[hvlines,cell-space-limits=3pt]
\CodeBefore
  \ShowCellNames
\Body
  \Block{2-2}{ } & & & test \\
  & & & blabla \\
  & & & some text & nothing
\end{NiceTabular}
```

1-1	1-2	test
2-1	2-2	blabla
3-1	some text	nothing

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

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

```


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


```

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

14.3 Default column type in `{NiceMatrix}`

New 6.11 The environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.) and the commande `\pAutoNiceMatrix` (and its variants) provide an option `columns-type` to specify the type of column which will be used (the initial value is, of course, `c`). The keys `l` and `r` are shortcuts for `columns-type=l` and `columns-type=r`.

```


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


```

The key `columns-type` is available in `\NiceMatrixOptions` but with the prefix `matrix`, which means that its name is, within `\NiceMatrixOptions` : `matrix/columns-type`.

14.4 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`.⁴³

```

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

```

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

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

```

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

```

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

⁴³It can also be used in `\RowStyle` (cf. p. 18).

14.5 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.

14.6 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 column⁴⁴. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 14) and in the `\CodeAfter` (cf. p. 28), `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`.

⁴⁴We 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}}}`

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

14.7 The key `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.⁴⁵

14.8 Color of the delimiters

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

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

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

This colour also applies to the delimiters drawn by the command `\SubMatrix` (cf. p. 29).

14.9 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.

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

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

⁴⁵The 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.

14.10 The command `\OnlyMainNiceMatrix`

The command `\OnlyMainNiceMatrix` executes its argument only when it is in the main part of the array, that is to say it is not in one of the exterior rows. If it is used outside an environment of `nicematrix`, that command is no-op.

For an example of utilisation, see tex.stackexchange.com/questions/488566

15 Use of Tikz with `nicematrix`

15.1 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`.⁴⁶

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.

```


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


```

The code to generate the matrix with a circled 5 is:

```


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


```

The code to generate the matrix with a circled 5 and a circle around the cell (2,2) is:

```


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


```

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).

```


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


```

The code to generate the matrix with a circled 5 and a circle around the cell (2,2) is:

```


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


```

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 55).

⁴⁶One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 23) and the computation of the “corners” (cf. p. 10).

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

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`.

15.1.1 The columns V of 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. In the following example, the command `\lipsum` is provided by the eponymous package.

```
\begin{NiceTabular}{V{10cm}}
\bfseries \large
Titre \\\
\lipsum[1][1-4]
\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.

We have used the nodes corresponding to the position of the potential rules, which are described below (cf. p. 43).

15.2 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`.⁴⁷

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

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

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

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.⁴⁸

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

⁴⁷There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

⁴⁸There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 21).

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`.⁴⁹

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

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 (with-out 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. 14). 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 array by using informations written on the `aux` file and created a second time during the contruction of the array itself).

Here is an example which uses these nodes in the `\CodeAfter`.

```
\begin{NiceArray}{c@{\;}c@{\;}c@{\;}c@{\;}c}[create-medium-nodes]
u_1 & -& u_0 & = & r & \\
u_2 & -& u_1 & = & r & \\
\end{NiceArray}
```

⁴⁹The 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.

```

u_3 &-& u_2 &=& r      \\
u_4 &-& u_3 &=& r      \\
\phantom{u_5} & & \phantom{u_4} & \smash{\vdots} & \\
u_n &-& u_{n-1} &=& r \\[3pt]
\hline
u_n &-& u_0 &=& nr \\
\CodeAfter
\tikz[very thick, red, opacity=0.4,name suffix = -medium]
\draw (1-1.north west) -- (2-3.south east)
(2-1.north west) -- (3-3.south east)
(3-1.north west) -- (4-3.south east)
(4-1.north west) -- (5-3.south east)
(5-1.north west) -- (6-3.south east) ;
\end{NiceArray}

```

$$\begin{array}{rcl}
u_1 - u_0 & = & r \\
u_2 - u_1 & = & r \\
u_3 - u_2 & = & r \\
u_4 - u_3 & = & r \\
& \vdots & \\
u_n - u_{n-1} & = & r \\
\hline
u_n - u_0 & = & nr
\end{array}$$

15.3 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`.

$\bullet^{1.5}$	\bullet^2	tulipe	lys
arum	$\bullet^{2.5}$	\bullet^3	violette mauve
muguet	dahlia	$\bullet^{3.5}$	\bullet^4

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}$

```

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

15.4 The nodes corresponding to the command `\SubMatrix`

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 29.

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}\}`).

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

16 API for the developers

The package `nicematrix` provides two variables which are internal but public⁵⁰:

- `\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).

⁵⁰According 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.

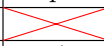
Example : We want to write a command `\crossbox` to draw a cross in the current cell. This command will take in an optional argument between square brackets for a list of pairs *key-value* which will be given to Tikz before the drawing.
It's possible to program such command `\crossbox` as follows, explicitly 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 { ! O { } }
{
  \tl_gput_right:Nx \g_nicematrix_code_after_tl
  {
    \__pantigny_crossbox:nnn
    { \int_use:c { c@iRow } }
    { \int_use:c { c@jCol } }
    { \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

17 Technical remarks

17.1 Diagonal lines

By default, all the diagonal lines⁵¹ of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That's why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

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

Example with parallelization (default):

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

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

⁵¹We 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]`.

17.2 The “empty” cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides. When the key `corners` is used (cf. p. 10), `nicematrix` computes corners consisting of empty cells. 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`. On the other side, a cell of a column of type `V` of `varwidth` (cf. p. 20) is empty when its TeX content has a width equal to zero.

17.3 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 idea⁵². The environment `{matrix}`

⁵²In 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).*

of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`⁵³. 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).

17.4 Incompatibilities

The package `nicematrix` is not compatible with the class `ieeeaccess` (because that class is not compatible with PGF/Tikz).⁵⁴

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}
```

The package `nicematrix` is not fully compatible with the package `arydshln` (because this package redefines many internal of `array`). By any means, in the context of `nicematrix`, it's recommended to draw dashed rules with the tools provided by `nicematrix`, by creating a customized line style with `custom-line`: cf. p. 11.

18 Examples

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

The key `tikz` of the command `\Block` is available only when Tikz is loaded.⁵⁵ For the following example, we 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} \
  \Block[tikz={pattern = sixpointed stars, pattern color = blue!15}]{ }
  {pattern = sixpointed stars, \ pattern color = blue!15}
```

⁵³And 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.

⁵⁴See <https://tex.stackexchange.com/questions/528975/error-loading-tikz-in-ieeeaccess-class>

⁵⁵By default, `nicematrix` only loads PGF, which is a sub-layer of Tikz.

```
& \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>	

18.2 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 12 p. 32.

Let's consider that we wish to number the notes of a tabular with stars.⁵⁶

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 ⁵⁷

```
\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 \\\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.}
& Mathilde & 23 mai 1988 \\\
```

⁵⁶Of course, it's realistic only when there is very few notes in the tabular.

⁵⁷In fact: the value of its argument.


```

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

```

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

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

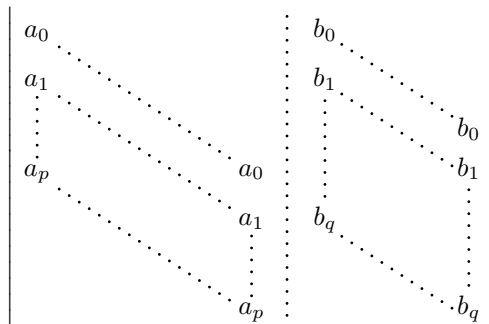
18.3 Dotted lines

An example with the resultant of two polynoms:

```

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

```



An example for a linear system:

```

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

```

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

18.4 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}}
\[\begin{pNiceMatrix}[last-row,last-col,nullify-dots,xdots/line-style={dashed,blue}]
1&&&\Vdots&&&\Vdots&\backslash
&\Ddots[line-style=standard]&\backslash
&&1&\backslash
&\Cdots[color=blue,line-style=dashed]&&&\blue 0&
&\Cdots&&&\blue 1&&&\Cdots&&\blue \leftarrow i&\backslash
&&&&1&\backslash
&&&\Vdots&&\Ddots[line-style=standard]&&&\Vdots&\backslash
&&&&&1&\backslash
&\Cdots&&&\blue 1&&\Cdots&&\Cdots&&\blue 0&&&\Cdots&&\blue \leftarrow j&\backslash
&&&&&&1&\backslash
&&&&&&&\Ddots[line-style=standard]&\backslash
&&&\Vdots&&&&\Vdots&&&1&\backslash
&&&\blue \overset{\uparrow}{i}&&&&\blue \overset{\uparrow}{j}&\backslash
\end{pNiceMatrix}\]
```

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

In fact, it's even possible to draw solid lines with the commands `\Cdots`, `\Vdots`, etc.⁵⁸

```
\NiceMatrixOptions
{nullify-dots,code-for-first-col = \color{blue},code-for-first-row=\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
```

⁵⁸In this document, the Tikz library `arrows.meta` has been loaded, which impacts the shape of the arrow tips.

```

& 1 & 1 & 1 & 1 & \Ldots & 1
\end{pNiceMatrix}$

```

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

18.5 Dashed rules

In the following example, we use the command `\Block` to draw dashed rules. For that example, Tikz should be loaded (by `\usepackage{tikz}`).

```

\begin{pNiceMatrix}
\Block[borders={bottom,right,tikz=dashed}]{2-2}{
1 & 2 & 0 & 0 & 0 & 0 \\
4 & 5 & 0 & 0 & 0 & 0 \\
0 & 0 & \Block[borders={bottom,top,right,left,tikz=dashed}]{2-2}{
7 & 1 & 0 & 0 \\
0 & 0 & -1 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & \Block[borders={left,top,tikz=dashed}]{2-2}{
3 & 4 \\
0 & 0 & 0 & 0 & 1 & 4
}
}
\end{pNiceMatrix}

```

$$\left(\begin{array}{cc|cc|cc} 1 & 2 & 0 & 0 & 0 & 0 \\ 4 & 5 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 7 & 1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{array} \right)$$

18.6 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}

```

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

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

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

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

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{pNiceArray}$`

...

`\end{NiceMatrixBlock}`

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

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

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

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

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

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

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

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

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

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

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.

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

```

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

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

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

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

18.7 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 block⁵⁹).

```

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

```

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

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don't spread the cells of the array. We recall that, on the other side, the commands `\hline` and `\Hline`, the specifier “|” and the options `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` spread the cells.⁶⁰

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).

```

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

```

⁵⁹We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

⁶⁰For the command `\cline`, see the remark p. 8.

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

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.

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}}
```

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

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

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”.

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

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

18.8 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`.

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

```
\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-last})
\SubMatrix({7-2}{last-6})
\SubMatrix({7-7}{last-last})
\begin{tikzpicture}
```

```

\mode [highlight = (9-2) (9-6)] { } ;
\mode [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 & & \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}\}

```

19 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.

```

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

```

```

11 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
12 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
13 \cs_generate_variant:Nn \@@_error:nn { n x }
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 } }

```

With Overleaf, a document is compiled in non-stop mode. When there is an error, there is no way to the user to use the key H in order to have more information. That’s why we decide to put that piece of information (for the messages with such information) in the main part of the message when the key `messages-for-Overleaf` is used (at load-time).

```

18 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
19 {
20   \bool_if:NTF \c_@@_messages_for_Overleaf_bool
21     { \msg_new:nnn { nicematrix } { #1 } { #2 } { #3 } }
22     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
23 }

```

We try to detect whether the compilation is done on Overleaf. We use `\c_sys_jobname_str` because, with Overleaf, the value of `\c_sys_jobname_str` is always “output”.

```

24 \str_if_eq:VnT \c_sys_jobname_str { output }
25 { \bool_set_true:N \c_@@_messages_for_Overleaf_bool }

```

```

26 \cs_new_protected:Npn \@@_msg_redirect_name:nn
27 { \msg_redirect_name:nnn { nicematrix } }

```

Technical definitions

```

28 \tl_new:N \l_@@_argspec_tl
29 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
30 \cs_generate_variant:Nn \keys_define:nn { n x }

31 \hook_gput_code:nnn { begindocument } { . }
32 {
33   \@@ifpackageloaded { varwidth }
34     { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_true_bool } }
35     { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_false_bool } }
36   \@@ifpackageloaded { arydshln }
37     { \bool_const:Nn \c_@@_arydshln_loaded_bool { \c_true_bool } }
38     { \bool_const:Nn \c_@@_arydshln_loaded_bool { \c_false_bool } }
39   \@@ifpackageloaded { booktabs }
40     { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_true_bool } }
41     { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_false_bool } }
42   \@@ifpackageloaded { enumitem }
43     { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_true_bool } }
44     { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_false_bool } }
45   \@@ifpackageloaded { tabularx }
46     { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_true_bool } }
47     { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_false_bool } }
48   { }
49   \@@ifpackageloaded { tikz }
50   {

```

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.

```

51     \bool_const:Nn \c_@@_tikz_loaded_bool \c_true_bool
52     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
53     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
54   }
55   {
56     \bool_const:Nn \c_@@_tikz_loaded_bool \c_false_bool
57     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
58     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
59   }
60 }

```

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 2022, the current version `revtex4-2` is 4.2e (compatible with `booktabs`).

```

61 \@@ifclassloaded { revtex4-1 }
62 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
63 {
64   \@@ifclassloaded { revtex4-2 }
65   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
66   {

```

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

```

67     \cs_if_exist:NT \rvtx@ifformat@geq
68     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
69     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
70   }
71 }

```

```

72 \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.

```

73 \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.

```

74 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
75 {
76   \iow_now:Nn \@mainaux
77   {
78     \ExplSyntaxOn
79     \cs_if_free:NT \pgfsyspdfmark
80     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
81     \ExplSyntaxOff
82   }
83   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
84 }

```

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.

```

85 \ProvideDocumentCommand \iddots { }
86 {
87   \mathinner
88   {
89     \tex_mkern:D 1 mu
90     \box_move_up:nn { 1 pt } { \hbox:n { . } }

```

```

91     \tex_mkern:D 2 mu
92     \box_move_up:nn { 4 pt } { \hbox:n { . } }
93     \tex_mkern:D 2 mu
94     \box_move_up:nn { 7 pt }
95     { \vbox:n { \kern 7 pt \hbox:n { . } } }
96     \tex_mkern:D 1 mu
97   }
98 }

```

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.

```

99 \hook_gput_code:nnn { begindocument } { . }
100 {
101   \@ifpackageloaded { booktabs }
102   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
103   { }
104 }
105 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
106 {
107   \cs_set_eq:NN \@_old_pgful@check@rerun \pgfutil@check@rerun

```

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

```

108   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
109   {
110     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
111     { \@_old_pgful@check@rerun { ##1 } { ##2 } }
112   }
113 }

```

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

```

114 \bool_new:N \l_@@_colortbl_loaded_bool
115 \hook_gput_code:nnn { begindocument } { . }
116 {
117   \@ifpackageloaded { colortbl }
118   { \bool_set_true:N \l_@@_colortbl_loaded_bool }
119   {

```

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.

```

120   \cs_set_protected:Npn \CT@arc@ { }
121   \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
122   \cs_set:Npn \CT@arc@ #1 #2
123   {
124     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
125     { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
126   }

```

Idem for `\CT@drs@`.

```

127   \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
128   \cs_set:Npn \CT@drs@ #1 #2
129   {
130     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
131     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
132   }
133   \cs_set:Npn \hline
134   {
135     \noalign { \ifnum 0 = ` } \fi
136     \cs_set_eq:NN \hskip \vskip
137     \cs_set_eq:NN \vrule \hrule
138     \cs_set_eq:NN \@width \@height

```

```

139         { \CT@arc@ \vline }
140         \futurelet \reserved@a
141         \@xhline
142     }
143 }
144 }

```

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.

```

145 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
146 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
147 {
148     \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
149     \int_compare:nNnT { #1 } > 1 { \multispan { \int_eval:n { #1 - 1 } } & }
150     \multispan { \int_eval:n { #2 - #1 + 1 } }
151     {
152         \CT@arc@
153         \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`⁶¹

```

154     \skip_horizontal:N \c_zero_dim
155 }

```

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`.

```

156 \everycr { }
157 \cr
158 \noalign { \skip_vertical:N -\arrayrulewidth }
159 }

```

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.

```

160 \cs_set:Npn \@@_cline

```

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

```

161 { \@@_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*.

```

162 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
163 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
164 {
165     \tl_if_empty:nTF { #3 }
166     { \@@_cline_iii:w #1|#2-#2 \q_stop }
167     { \@@_cline_ii:w #1|#2-#3 \q_stop }
168 }
169 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
170 { \@@_cline_iii:w #1|#2-#3 \q_stop }
171 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
172 {

```

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).

```

173     \int_compare:nNnT { #1 } < { #2 }
174     { \multispan { \int_eval:n { #2 - #1 } } & }
175     \multispan { \int_eval:n { #3 - #2 + 1 } }
176     {
177         \CT@arc@

```

⁶¹See question 99041 on TeX StackExchange.

```

178     \leaders \hrule \@height \arrayrulewidth \hfill
179     \skip_horizontal:N \c_zero_dim
180 }

```

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

```

181 \peek_meaning_remove_ignore_spaces:NTF \cline
182 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
183 { \everycr { } \cr }
184 }
185 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

The following command is a small shortcut.

```

186 \cs_new:Npn \@@_math_toggle_token:
187 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

```

```

188 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
189 {
190   \tl_if_blank:nF { #1 }
191   {
192     \tl_if_head_eq_meaning:nNTF { #1 } [
193       { \cs_set:Npn \CT@arc@ { \color #1 } }
194       { \cs_set:Npn \CT@arc@ { \color { #1 } } }
195     ]
196   }
197 \cs_generate_variant:Nn \@@_set_CT@arc@:n { V }

```

```

198 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
199 {
200   \tl_if_head_eq_meaning:nNTF { #1 } [
201     { \cs_set:Npn \CT@drsc@ { \color #1 } }
202     { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
203   ]
204 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { V }

```

The following command must *not* be protected since it will be used to write instructions in the (internal) `\CodeBefore`.

```

205 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
206 {
207   \tl_if_head_eq_meaning:nNTF { #2 } [
208     { #1 #2 }
209     { #1 { #2 } }
210   ]
211 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N V }

```

The following command must be protected because of its use of the command `\color`.

```

212 \cs_new_protected:Npn \@@_color:n #1
213 {
214   \tl_if_blank:nF { #1 }
215   { \@@_exp_color_arg:Nn \color { #1 } }
216 }
217 \cs_generate_variant:Nn \@@_color:n { V }

```

```

218 \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`.

```

219 \bool_new:N \l_@@_siunitx_loaded_bool
220 \hook_gput_code:nnn { begindocument } { . }
221 {
222   \@ifpackageloaded { siunitx }
223   { \bool_set_true:N \l_@@_siunitx_loaded_bool }

```

```

224     { }
225 }

```

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

```

226 \hook_gput_code:nnn { begindocument } { . }
227 {
228   \bool_if:nTF { ! \l_@@_siunitx_loaded_bool }
229   { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
230   {
231     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
232     {
233       \renewcommand*{\NC@rewrite@S}[1] []
234       {

```

`\@temptokena` is a `toks` (not supported by the L3 programming layer).

```

235       \@temptokena \exp_after:wN
236       { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
237       \NC@find
238     }
239   }
240 }
241 }

```

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.

```

242 \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).

```

243 \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.

```

244 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
245 { \int_use:N \g_@@_env_int }

```

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

```

246 \cs_new_protected:Npn \@@_qpoint:n #1
247 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

248 \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).

```

249 \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.).

```
250 \dim_new:N \l_@@_col_width_dim
251 \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.

```
252 \int_new:N \g_@@_row_total_int
253 \int_new:N \g_@@_col_total_int
```

The following parameter will be used by `\@@_create_row_node`: to avoid to create the same row-node twice (at the end of the array).

```
254 \int_new:N \g_@@_last_row_node_int
```

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

```
255 \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.

```
256 \str_new:N \l_@@_hpos_cell_str
257 \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`.

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

Idem for the mono-row blocks.

```
259 \dim_new:N \g_@@_blocks_ht_dim
260 \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}`).

```
261 \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`.

```
262 \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.

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

The following key corresponds to the key `notes/detect_duplicates`.

```
264 \bool_new:N \l_@@_notes_detect_duplicates_bool
265 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

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

```
266 \bool_new:N \g_@@_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.

```
267 \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.

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

The following dimension will be used for the total width of composite rules (*total* means that the spaces on both sides are included).

```
269 \dim_new:N \l_@@_rule_width_dim
```

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

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

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

```
271 \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.

```
272 \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 }`).

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

The following parameter corresponds to the key `columns-type` of the environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. and also the key `matrix / columns-type` of `\NiceMatrixOptions`. However, it does *not* contain the value provided by the final user. Indeed, a transformation is done in order to have a preamble (for the package `array`) which is `nicematrix`-aware. That transformation is done with the command `\@@_set_preamble:Nn`.

```
274 \tl_new:N \l_@@_columns_type_tl
275 \hook_gput_code:nnn { begindocument } { . }
276 { \@@_set_preamble:Nn \l_@@_columns_type_tl { c } }
```

```
277 \cs_new_protected:Npn \@@_test_if_math_mode:
278 {
279   \if_mode_math: \else:
280     \@@_fatal:n { Outside-math-mode }
281   \fi:
282 }
```

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

```
283 \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.

```
284 \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”.

```
285 \colorlet { nicematrix-last-col } { . }
286 \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`).

```
287 \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*.

```
288 \tl_new:N \g_@@_com_or_env_str
289 \tl_gset: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).

```
290 \cs_new:Npn \@@_full_name_env:
291 {
292   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
293   { command \space \c_backslash_str \g_@@_name_env_str }
294   { environment \space \{ \g_@@_name_env_str \} }
295 }
```

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*.

```
296 \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.

```
297 \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.

```
298 \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.

```
299 \int_new:N \l_@@_old_iRow_int
300 \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 sequence will contain the names (without backslash) of the commands created by `custom-line` (commands used by the final user in order to draw horizontal rules).

```
301 \seq_new:N \l_@@_custom_line_commands_seq
```

The following token list corresponds to the key `rules/color` available in the environments.

```
302 \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.

```
303 \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.

```
304 \bool_new:N \l_@@_X_columns_aux_bool
305 \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.

```
306 \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).

```
307 \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).

```
308 \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`).

```
309 \tl_new:N \l_@@_code_before_tl
310 \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).

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

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

```
312 \dim_new:N \l_@@_x_initial_dim
313 \dim_new:N \l_@@_y_initial_dim
314 \dim_new:N \l_@@_x_final_dim
315 \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.

```
316 \dim_zero_new:N \l_@@_tmpc_dim
317 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
318 \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”.

```
319 \dim_new:N \g_@@_width_last_col_dim
320 \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.

```
321 \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!).

```
322 \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}`.

```
323 \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.

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

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

```
325 \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`).

```
326 \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.

```
327 \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).

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

```
329 \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`).

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

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

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

```
333 \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 `\CodeBefore`. 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.

```
334 \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).

```
335 \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`.

```
336 \tl_new:N \l_@@_fill_tl
337 \tl_new:N \l_@@_draw_tl
338 \seq_new:N \l_@@_tikz_seq
339 \clist_new:N \l_@@_borders_clist
340 \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` and also the key `color` of the command `\RowStyle`.

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

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

```
342 \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`).

```
343 \str_new:N \l_@@_hpos_block_str
344 \str_set:Nn \l_@@_hpos_block_str { c }
345 \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`.

```
346 \tl_new:N \l_@@_vpos_of_block_tl
347 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

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

The following flag corresponds to the keys `vlines` and `hlines` of the command `\Block` (the key `hvlines` is the conjunction of both).

```
349 \bool_new:N \l_@@_vlines_block_bool
350 \bool_new:N \l_@@_hlines_block_bool
```

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

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

352 \dim_new:N \l_@@_submatrix_extra_height_dim
353 \dim_new:N \l_@@_submatrix_left_xshift_dim
354 \dim_new:N \l_@@_submatrix_right_xshift_dim
355 \clist_new:N \l_@@_hlines_clist
356 \clist_new:N \l_@@_vlines_clist
357 \clist_new:N \l_@@_submatrix_hlines_clist
358 \clist_new:N \l_@@_submatrix_vlines_clist
```

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

```
359 \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.

```
360 \int_new:N \l_@@_first_row_int
361 \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.

```
362 \int_new:N \l_@@_first_col_int
363 \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).

```
364 \int_new:N \l_@@_last_row_int
365 \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”.⁶²

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

Idem for `\l_@@_last_col_without_value_bool`

```
367 \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.

```
368 \int_new:N \l_@@_last_col_int
369 \int_set:Nn \l_@@_last_col_int { -2 }
```

⁶²We can’t use `\l_@@_last_row_int` for this usage because, if `nicematrix` has read its value from the `aux` file, the value of the counter won’t be `-1` any longer.

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

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

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

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

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

Some utilities

```
371 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
372 {
373   \tl_set:Nn \l_tmpa_tl { #1 }
374   \tl_set:Nn \l_tmpb_tl { #2 }
375 }
```

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.

```
376 \cs_new_protected:Npn \@@_expand_clist:N #1
377 {
378   \clist_if_in:NnF #1 { all }
379   {
380     \clist_clear:N \l_tmpa_clist
381     \clist_map_inline:Nn #1
382     {
383       \tl_if_in:nnTF { ##1 } { - }
384       { \@@_cut_on_hyphen:w ##1 \q_stop }
385       {
386         \tl_set:Nn \l_tmpa_tl { ##1 }
387         \tl_set:Nn \l_tmpb_tl { ##1 }
388       }
389       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
390       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
391     }
392     \tl_set_eq:NN #1 \l_tmpa_clist
393   }
394 }
```

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.

```
395 \newcounter { tablarnote }
```

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

```
396 \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.

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



```
398 \seq_new:N \l_@@_notes_labels_seq
```

```
399 \newcounter{nicematrix_draft}
400 \cs_new_protected:Npn \@@_notes_format:n #1
401 {
402   \setcounter { nicematrix_draft } { #1 }
403   \@@_notes_style:n { nicematrix_draft }
404 }
```

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

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

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

```
406 \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`.

```
407 \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`.

```
408 \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`).

```
409 \hook_gput_code:nnn { begindocument } { . }
410 {
411   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
412   {
413     \NewDocumentCommand \tabularnote { m }
414     { \@@_error:n { enumitem-not-loaded } }
415   }
416   {
```

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.

```
417   \newlist { tabularnotes } { enumerate } { 1 }
418   \setlist [ tabularnotes ]
419   {
420     topsep = 0pt ,
421     noitemsep ,
422     leftmargin = * ,
423     align = left ,
424     labelsep = 0pt ,
425     label =
426     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
427   }
428   \newlist { tabularnotes* } { enumerate* } { 1 }
429   \setlist [ tabularnotes* ]
430   {
431     afterlabel = \nobreak ,
432     itemjoin = \quad ,
433     label =
434     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
435   }
```

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).

```

436 \NewDocumentCommand \tabularnote { m }
437 {
438   \bool_if:nTF { ! \g_@@_NiceArray_bool && \l_@@_in_env_bool }
439     { \@@_error:n { tabularnote~forbidden } }
440     {

```

You have to see whether the argument of `\tabularnote` has yet been used as argument of another `\tabularnote` in the same tabular. In that case, there will be only one note (for both commands `\tabularnote`) at the end of the tabular. We search the argument of our command `\tabularnote` in the `\g_@@_tabularnotes_seq`. The position in the sequence will be stored in `\l_tmpa_int` (0 if the text is not in the sequence yet).

```

441   \int_zero:N \l_tmpa_int
442   \bool_if:NT \l_@@_notes_detect_duplicates_bool
443     {
444     \seq_map_indexed_inline:Nn \g_@@_tabularnotes_seq
445       {
446         \tl_if_eq:nnT { #1 } { ##2 }
447         { \int_set:Nn \l_tmpa_int { ##1 } \seq_map_break: }
448       }
449     }
450   \int_compare:nNnTF \l_tmpa_int = 0
451     {
452     \stepcounter { tabularnote }
453     \seq_put_right:Nx \l_@@_notes_labels_seq
454       { \@@_notes_format:n { \int_use:c { c @ tabularnote } } }
455     \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
456   }
457   {
458   \seq_put_right:Nx \l_@@_notes_labels_seq
459     { \@@_notes_format:n { \int_use:N \l_tmpa_int } }
460   }
461   \peek_meaning:NF \tabularnote
462   {

```

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.

```

463   \hbox_set:Nn \l_tmpa_box
464   {

```

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

```

465   \@@_notes_label_in_tabular:n
466   {
467     \seq_use:Nnnn
468       \l_@@_notes_labels_seq { , } { , } { , }
469   }
470 }

```

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.

```

471   \addtocounter { tabularnote } { -1 }
472   \refstepcounter { tabularnote }
473   \seq_clear:N \l_@@_notes_labels_seq
474   \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).

```

475         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
476     }
477 }
478 }
479 }
480 }
```

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.

```

481 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
482 {
483     \begin { pgfscope }
484     \pgfset
485     {
486         outer~sep = \c_zero_dim ,
487         inner~sep = \c_zero_dim ,
488         minimum~size = \c_zero_dim
489     }
490     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
491     \pgfnode
492     { rectangle }
493     { center }
494     {
495         \vbox_to_ht:nn
496         { \dim_abs:n { #5 - #3 } }
497         {
498             \vfill
499             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
500         }
501     }
502     { #1 }
503     { }
504     \end { pgfscope }
505 }
```

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.

```

506 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
507 {
508     \begin { pgfscope }
509     \pgfset
510     {
511         outer~sep = \c_zero_dim ,
512         inner~sep = \c_zero_dim ,
513         minimum~size = \c_zero_dim
514     }
515     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
516     \pgfpointdiff { #3 } { #2 }
517     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
518     \pgfnode
519     { rectangle }
520     { center }
521     {
522         \vbox_to_ht:nn
```

```

523         { \dim_abs:n \l_tmpb_dim }
524         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
525     }
526     { #1 }
527     { }
528 \end { pgfscope }
529 }

```

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.

```

530 \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 `\arrayrulewidth`. It's possible to disable this feature with the key `\l_@@_standard_line_bool`.

```

531 \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`).

```

532 \dim_new:N \l_@@_cell_space_top_limit_dim
533 \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.

```

534 \dim_new:N \l_@@_xdots_inter_dim
535 \hook_gput_code:nnn { begindocument } { . }
536 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }

```

We use a hook only by 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).

```

537 \dim_new:N \l_@@_xdots_shorten_start_dim
538 \dim_new:N \l_@@_xdots_shorten_end_dim
539 \hook_gput_code:nnn { begindocument } { . }
540 {
541     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
542     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
543 }

```

We use a hook only by 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.

```

544 \dim_new:N \l_@@_xdots_radius_dim
545 \hook_gput_code:nnn { begindocument } { . }
546 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }

```

We use a hook only by security in case `revtex4-1` is used (even though 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.

```
547 \tl_new:N \l_@@_xdots_line_style_tl
548 \tl_const:Nn \c_@@_standard_tl { standard }
549 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
550 \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).

```
551 \tl_new:N \l_@@_baseline_tl
552 \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`).

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

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

```
554 \bool_new:N \l_@@_parallelize_diags_bool
555 \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.

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

```
557 \dim_new:N \l_@@_notes_above_space_dim
558 \hook_gput_code:nnn { begindocument } { . }
559 { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

We use a hook only by security in case `revtex4-1` is used (even though 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.).

```
560 \bool_new:N \l_@@_nullify_dots_bool
```

The following flag corresponds to the key `respect-arraystretch` (that key has an effect on the blocks).

```
561 \bool_new:N \l_@@_respect_arraystretch_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).

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

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

```
563 \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.

```
564 \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”.

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

```
566 \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).

```
567 \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`).

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

```
569 \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`.

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

```
571 \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.

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

```
573 \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 “:”.

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

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

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

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

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

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

```
578 {
```

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

```
580   {
```

```
581     \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.

```

582     { \cs_if_exist_p:N \tikzpicture }
583     { \str_if_eq_p:nn { #1 } { standard } }
584     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
585     { \@@_error:n { bad-option-for-line-style } }
586   } ,
587   line-style .value_required:n = true ,
588   color .tl_set:N = \l_@@_xdots_color_tl ,
589   color .value_required:n = true ,
590   shorten .code:n =
591     \hook_gput_code:nnn { begindocument } { . }
592     {
593       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
594       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
595     } ,
596   shorten-start .code:n =
597     \hook_gput_code:nnn { begindocument } { . }
598     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
599   shorten-end .code:n =
600     \hook_gput_code:nnn { begindocument } { . }
601     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,

```

We use a hook only by security in case `revtex4-1` is used (even though it is obsolete). Idem for the following keys.

```

602   shorten .value_required:n = true ,
603   shorten-start .value_required:n = true ,
604   shorten-end .value_required:n = true ,
605   radius .code:n =
606     \hook_gput_code:nnn { begindocument } { . }
607     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
608   radius .value_required:n = true ,
609   inter .code:n =
610     \hook_gput_code:nnn { begindocument } { . }
611     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
612   radius .value_required:n = true ,

```

The options `down` and `up` are not documented for the final user because he should use the syntax with `^` and `_`.

```

613   down .tl_set:N = \l_@@_xdots_down_tl ,
614   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).

```

615   draw-first .code:n = \prg_do_nothing: ,
616   unknown .code:n = \@@_error:n { Unknown-key-for-~xdots }
617 }

```

```

618 \keys_define:nn { NiceMatrix / rules }
619 {
620   color .tl_set:N = \l_@@_rules_color_tl ,
621   color .value_required:n = true ,
622   width .dim_set:N = \arrayrulewidth ,
623   width .value_required:n = true ,
624   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
625 }

```

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

```

626 \keys_define:nn { NiceMatrix / Global }
627 {

```

```

628 custom-line .code:n = \l_@@_custom_line:n { #1 } ,
629 delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
630 delimiters .value_required:n = true ,
631 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
632 rules .value_required:n = true ,
633 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
634 standard-cline .default:n = true ,
635 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
636 cell-space-top-limit .value_required:n = true ,
637 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
638 cell-space-bottom-limit .value_required:n = true ,
639 cell-space-limits .meta:n =
640 {
641     cell-space-top-limit = #1 ,
642     cell-space-bottom-limit = #1 ,
643 } ,
644 cell-space-limits .value_required:n = true ,
645 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
646 light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
647 light-syntax .default:n = true ,
648 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
649 end-of-row .value_required:n = true ,
650 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
651 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
652 last-row .int_set:N = \l_@@_last_row_int ,
653 last-row .default:n = -1 ,
654 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
655 code-for-first-col .value_required:n = true ,
656 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
657 code-for-last-col .value_required:n = true ,
658 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
659 code-for-first-row .value_required:n = true ,
660 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
661 code-for-last-row .value_required:n = true ,
662 hlines .clist_set:N = \l_@@_hlines_clist ,
663 vlines .clist_set:N = \l_@@_vlines_clist ,
664 hlines .default:n = all ,
665 vlines .default:n = all ,
666 vlines-in-sub-matrix .code:n =
667 {
668     \tl_if_single_token:nTF { #1 }
669     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
670     { \@@_error:n { One~letter~allowed } }
671 } ,
672 vlines-in-sub-matrix .value_required:n = true ,
673 hvlines .code:n =
674 {
675     \clist_set:Nn \l_@@_vlines_clist { all }
676     \clist_set:Nn \l_@@_hlines_clist { all }
677 } ,
678 hvlines-except-borders .code:n =
679 {
680     \clist_set:Nn \l_@@_vlines_clist { all }
681     \clist_set:Nn \l_@@_hlines_clist { all }
682     \bool_set_true:N \l_@@_except_borders_bool
683 } ,
684 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.

```

685 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
686 renew-dots .value_forbidden:n = true ,
687 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,

```



```

688 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
689 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
690 create-extra-nodes .meta:n =
691   { create-medium-nodes , create-large-nodes } ,
692 left-margin .dim_set:N = \l_@@_left_margin_dim ,
693 left-margin .default:n = \arraycolsep ,
694 right-margin .dim_set:N = \l_@@_right_margin_dim ,
695 right-margin .default:n = \arraycolsep ,
696 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
697 margin .default:n = \arraycolsep ,
698 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
699 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
700 extra-margin .meta:n =
701   { extra-left-margin = #1 , extra-right-margin = #1 } ,
702 extra-margin .value_required:n = true ,
703 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
704 respect-arraystretch .default:n = true
705 }

```

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

```

706 \keys_define:nn { NiceMatrix / Env }
707   {

```

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

```

708   hvlines-except-corners .code:n = \@@_fatal:n { hvlines-except-corners } ,
709   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
710   corners .clist_set:N = \l_@@_corners_clist ,
711   corners .default:n = { NW , SW , NE , SE } ,
712   code-before .code:n =
713   {
714     \tl_if_empty:nF { #1 }
715     {
716       \tl_put_right:Nn \l_@@_code_before_tl { #1 }
717       \bool_set_true:N \l_@@_code_before_bool
718     }
719   } ,

```

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

```

720   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
721   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
722   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
723   baseline .tl_set:N = \l_@@_baseline_tl ,
724   baseline .value_required:n = true ,
725   columns-width .code:n =
726   { \tl_if_eq:nnTF { #1 } { auto }
727     { \bool_set_true:N \l_@@_auto_columns_width_bool }
728     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
729   columns-width .value_required:n = true ,
730   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.

```

731   \legacy_if:nF { measuring@ }
732   {
733     \str_set:Nn \l_tmpa_str { #1 }
734     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
735     { \@@_error:nn { Duplicate-name } { #1 } }
736     { \seq_gput_left:N \g_@@_names_seq \l_tmpa_str }
737     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
738   } ,
739   name .value_required:n = true ,

```

```

740 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
741 code-after .value_required:n = true ,
742 colortbl-like .code:n =
743   \bool_set_true:N \l_@@_colortbl_like_bool
744   \bool_set_true:N \l_@@_code_before_bool ,
745 colortbl-like .value_forbidden:n = true
746 }
747 \keys_define:nn { NiceMatrix / notes }
748 {
749   para .bool_set:N = \l_@@_notes_para_bool ,
750   para .default:n = true ,
751   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
752   code-before .value_required:n = true ,
753   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
754   code-after .value_required:n = true ,
755   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
756   bottomrule .default:n = true ,
757   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
758   style .value_required:n = true ,
759   label-in-tabular .code:n =
760     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
761   label-in-tabular .value_required:n = true ,
762   label-in-list .code:n =
763     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
764   label-in-list .value_required:n = true ,
765   enumitem-keys .code:n =
766     {
767       \hook_gput_code:nnn { begindocument } { . }
768       {
769         \bool_if:NT \c_@@_enumitem_loaded_bool
770         { \setlist* [ tabularnotes ] { #1 } }
771       }
772     } ,
773   enumitem-keys .value_required:n = true ,
774   enumitem-keys-para .code:n =
775     {
776       \hook_gput_code:nnn { begindocument } { . }
777       {
778         \bool_if:NT \c_@@_enumitem_loaded_bool
779         { \setlist* [ tabularnotes* ] { #1 } }
780       }
781     } ,
782   enumitem-keys-para .value_required:n = true ,
783   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
784   detect-duplicates .default:n = true ,
785   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
786 }
787 \keys_define:nn { NiceMatrix / delimiters }
788 {
789   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
790   max-width .default:n = true ,
791   color .tl_set:N = \l_@@_delimiters_color_tl ,
792   color .value_required:n = true ,
793 }

```

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

```

794 \keys_define:nn { NiceMatrix }
795 {
796   NiceMatrixOptions .inherit:n =
797     { NiceMatrix / Global } ,
798   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,

```

```

799 NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
800 NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
801 NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
802 NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
803 SubMatrix / rules .inherit:n = NiceMatrix / rules ,
804 CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
805 NiceMatrix .inherit:n =
806   {
807     NiceMatrix / Global ,
808     NiceMatrix / Env ,
809   } ,
810 NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
811 NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
812 NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
813 NiceTabular .inherit:n =
814   {
815     NiceMatrix / Global ,
816     NiceMatrix / Env
817   } ,
818 NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
819 NiceTabular / rules .inherit:n = NiceMatrix / rules ,
820 NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
821 NiceArray .inherit:n =
822   {
823     NiceMatrix / Global ,
824     NiceMatrix / Env ,
825   } ,
826 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
827 NiceArray / rules .inherit:n = NiceMatrix / rules ,
828 NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
829 pNiceArray .inherit:n =
830   {
831     NiceMatrix / Global ,
832     NiceMatrix / Env ,
833   } ,
834 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
835 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
836 pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
837 }

```

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

```

838 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
839 {
840   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
841   width .value_required:n = true ,
842   last-col .code:n = \tl_if_empty:nF { #1 }
843     { \@@_error:n { last-col~non~empty~for~NiceMatrixOptions } }
844     \int_zero:N \l_@@_last_col_int ,
845   small .bool_set:N = \l_@@_small_bool ,
846   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.

```

847 renew-matrix .code:n = \@@_renew_matrix: ,
848 renew-matrix .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}`.

```

849 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.

```

850 columns-width .code:n =
851   \tl_if_eq:nnTF { #1 } { auto }
852   { \@@_error:n { Option~auto~for~columns~width } }
853   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

854 allow-duplicate-names .code:n =
855   \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
856 allow-duplicate-names .value_forbidden:n = true ,
857 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
858 notes .value_required:n = true ,
859 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
860 sub-matrix .value_required:n = true ,
861 matrix / columns-type .code:n =
862   \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
863 matrix / columns-type .value_required:n = true ,
864 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
865 }

```

`\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.

```

866 \NewDocumentCommand \NiceMatrixOptions { m }
867 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

868 \keys_define:nn { NiceMatrix / NiceMatrix }
869 {
870   last-col .code:n = \tl_if_empty:nTF {#1}
871     {
872       \bool_set_true:N \l_@@_last_col_without_value_bool
873       \int_set:Nn \l_@@_last_col_int { -1 }
874     }
875     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
876   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
877   columns-type .value_required:n = true ,
878   l .meta:n = { columns-type = l } ,
879   r .meta:n = { columns-type = r } ,
880   small .bool_set:N = \l_@@_small_bool ,
881   small .value_forbidden:n = true ,
882   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
883 }

```

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

```

884 \keys_define:nn { NiceMatrix / NiceArray }
885 {

```

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.

```

886   small .bool_set:N = \l_@@_small_bool ,
887   small .value_forbidden:n = true ,
888   last-col .code:n = \tl_if_empty:nF { #1 }
889     { \@@_error:n { last~col~non~empty~for~NiceArray } }
890     { \int_zero:N \l_@@_last_col_int ,
891   notes / para .bool_set:N = \l_@@_notes_para_bool ,
892   notes / para .default:n = true ,

```

```

893 notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
894 notes / bottomrule .default:n = true ,
895 tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
896 tabularnote .value_required:n = true ,
897 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
898 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
899 unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
900 }
901 \keys_define:nn { NiceMatrix / pNiceArray }
902 {
903   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
904   last-col .code:n = \tl_if_empty:nF {#1}
905     { \@@_error:n { last-col~non~empty~for~NiceArray } }
906     \int_zero:N \l_@@_last_col_int ,
907   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
908   small .bool_set:N = \l_@@_small_bool ,
909   small .value_forbidden:n = true ,
910   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
911   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
912   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
913 }

```

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

```

914 \keys_define:nn { NiceMatrix / NiceTabular }
915 {

```

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.

```

916   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
917   \bool_set_true:N \l_@@_width_used_bool ,
918   width .value_required:n = true ,
919   notes / para .bool_set:N = \l_@@_notes_para_bool ,
920   notes / para .default:n = true ,
921   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
922   notes / bottomrule .default:n = true ,
923   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
924   tabularnote .value_required:n = true ,
925   last-col .code:n = \tl_if_empty:nF {#1}
926     { \@@_error:n { last-col~non~empty~for~NiceArray } }
927     \int_zero:N \l_@@_last_col_int ,
928   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
929   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
930   unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
931 }

```

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}).

```

932 \cs_new_protected:Npn \@@_cell_begin:w
933 {

```

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*).

```

934   \tl_gclear:N \g_@@_post_action_cell_tl

```

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

```
935 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
936 \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.

```
937 \int_compare:nNnT \c@jCol = 1
938 { \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).

```
939 \hbox_set:Nw \l_@@_cell_box
940 \bool_if:NF \l_@@_NiceTabular_bool
941 {
942   \c_math_toggle_token
943   \bool_if:NT \l_@@_small_bool \scriptstyle
944 }
```

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).

```
945 \color { nicematrix }
946 \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.

```
947 \int_compare:nNnTF \c@iRow = 0
948 {
949   \int_compare:nNnT \c@jCol > 0
950   {
951     \l_@@_code_for_first_row_tl
952     \xglobal \colorlet { nicematrix-first-row } { . }
953   }
954 }
955 {
956   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
957   {
958     \l_@@_code_for_last_row_tl
959     \xglobal \colorlet { nicematrix-last-row } { . }
960   }
961 }
962 }
```

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.

```
963 \cs_new_protected:Npn \@@_begin_of_row:
964 {
965   \int_gincr:N \c@iRow
966   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
967   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
968   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
969   \pgfpicture
970   \pgfrememberpicturepositiononpagetrue
971   \pgfcoordinate
972   { \@@_env: - row - \int_use:N \c@iRow - base }
```

```

973     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
974     \str_if_empty:NF \l_@@_name_str
975     {
976         \pgfnodealias
977         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
978         { \@@_env: - row - \int_use:N \c@iRow - base }
979     }
980     \endpgfpicture
981 }

```

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.

```

982 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
983 {
984     \int_compare:nNnTF \c@iRow = 0
985     {
986         \dim_gset:Nn \g_@@_dp_row_zero_dim
987         { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
988         \dim_gset:Nn \g_@@_ht_row_zero_dim
989         { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
990     }
991     {
992         \int_compare:nNnT \c@iRow = 1
993         {
994             \dim_gset:Nn \g_@@_ht_row_one_dim
995             { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
996         }
997     }
998 }
999 \cs_new_protected:Npn \@@_rotate_cell_box:
1000 {
1001     \box_rotate:Nn \l_@@_cell_box { 90 }
1002     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1003     {
1004         \vbox_set_top:Nn \l_@@_cell_box
1005         {
1006             \vbox_to_zero:n { }
1007             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1008             \box_use:N \l_@@_cell_box
1009         }
1010     }
1011     \bool_gset_false:N \g_@@_rotate_bool
1012 }
1013 \cs_new_protected:Npn \@@_adjust_size_box:
1014 {
1015     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1016     {
1017         \box_set_wd:Nn \l_@@_cell_box
1018         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1019         \dim_gzero:N \g_@@_blocks_wd_dim
1020     }
1021     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1022     {
1023         \box_set_dp:Nn \l_@@_cell_box
1024         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1025         \dim_gzero:N \g_@@_blocks_dp_dim
1026     }
1027     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim

```

```

1028     {
1029         \box_set_ht:Nn \l_@@_cell_box
1030         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1031         \dim_gzero:N \g_@@_blocks_ht_dim
1032     }
1033 }
1034 \cs_new_protected:Npn \@@_cell_end:
1035 {
1036     \@@_math_toggle_token:
1037     \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.

```

1038     \g_@@_post_action_cell_tl
1039     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1040     \@@_adjust_size_box:
1041     \box_set_ht:Nn \l_@@_cell_box
1042     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1043     \box_set_dp:Nn \l_@@_cell_box
1044     { \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”).

```

1045     \dim_gset:Nn \g_@@_max_cell_width_dim
1046     { \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”.

```

1047     \@@_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.

```

1048     \bool_if:NTF \g_@@_empty_cell_bool
1049     { \box_use_drop:N \l_@@_cell_box }
1050     {
1051         \bool_lazy_or:nnTF
1052         \g_@@_not_empty_cell_bool
1053         { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1054         \@@_node_for_cell:
1055         { \box_use_drop:N \l_@@_cell_box }
1056     }
1057     \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1058     \bool_gset_false:N \g_@@_empty_cell_bool
1059     \bool_gset_false:N \g_@@_not_empty_cell_bool
1060 }

```


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

```

1061 \cs_new_protected:Npn \@@_node_for_cell:
1062 {
1063   \pgfpicture
1064   \pgfsetbaseline \c_zero_dim
1065   \pgfrememberpicturepositiononpagetrue
1066   \pgfset
1067   {
1068     inner~sep = \c_zero_dim ,
1069     minimum~width = \c_zero_dim
1070   }
1071   \pgfnode
1072   { rectangle }
1073   { base }
1074   { \box_use_drop:N \l_@@_cell_box }
1075   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1076   { }
1077   \str_if_empty:NF \l_@@_name_str
1078   {
1079     \pgfnodealias
1080     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1081     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1082   }
1083   \endpgfpicture
1084 }

```

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.

```

1085 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1086 {
1087   \cs_new_protected:Npn \@@_patch_node_for_cell:
1088   {
1089     \hbox_set:Nn \l_@@_cell_box
1090     {
1091       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1092       \hbox_overlap_left:n
1093       {
1094         \pgfsys@markposition
1095         { \@@_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.

```

1096         #1
1097       }
1098       \box_use:N \l_@@_cell_box
1099       \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1100       \hbox_overlap_left:n
1101       {
1102         \pgfsys@markposition
1103         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1104         #1
1105       }
1106     }
1107   }
1108 }

```

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

```

1109 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1110 {
1111   \@@_patch_node_for_cell:n
1112   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }

```

```

1113 }
1114 { \@@_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.

```

1115 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1116 {
1117   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1118     { \g_@@_ #2 _ lines _ tl }
1119     {
1120       \use:c { @@ _ draw _ #2 : nnn }
1121       { \int_use:N \c@iRow }
1122       { \int_use:N \c@jCol }
1123       { \exp_not:n { #3 } }
1124     }
1125 }
1126 \cs_new_protected:Npn \@@_array:n
1127 {
1128   \bool_if:NTF \l_@@_NiceTabular_bool
1129     { \dim_set_eq:NN \col@sep \tabcolsep }
1130     { \dim_set_eq:NN \col@sep \arraycolsep }
1131   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1132     { \cs_set_nopar:Npn \@halignto { } }
1133     { \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`.

```

1134   \@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.
1135   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1136 }
1137 \cs_generate_variant:Nn \@@_array:n { V }

```

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.

```

1138 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1139 \cs_new_protected:Npn \@@_create_row_node:
1140 {
1141   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1142     {
1143       \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1144       \@@_create_row_node_i:
1145     }
1146 }

```

```

1147 \cs_new_protected:Npn \@@_create_row_node_i:
1148 {

```

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

```

1149   \hbox
1150   {
1151     \bool_if:NT \l_@@_code_before_bool
1152     {
1153       \vtop
1154       {
1155         \skip_vertical:N 0.5\arrayrulewidth
1156         \pgfsys@markposition
1157         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1158         \skip_vertical:N -0.5\arrayrulewidth
1159       }
1160     }
1161     \pgfpicture
1162     \pgfrememberpicturepositiononpagetrue
1163     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1164     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1165     \str_if_empty:NF \l_@@_name_str
1166     {
1167       \pgfnodealias
1168       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1169       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1170     }
1171     \endpgfpicture
1172   }
1173 }

```

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

```

1174 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1175 \cs_new_protected:Npn \@@_everycr_i:
1176 {
1177   \int_gzero:N \c@jCol
1178   \bool_gset_false:N \g_@@_after_col_zero_bool
1179   \bool_if:NF \g_@@_row_of_col_done_bool
1180   {
1181     \@@_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).

```

1182     \tl_if_empty:NF \l_@@_hlines_clist
1183     {
1184       \tl_if_eq:NnF \l_@@_hlines_clist { all }
1185       {
1186         \exp_args:NNx
1187         \clist_if_in:NnT
1188         \l_@@_hlines_clist
1189         { \int_eval:n { \c@iRow + 1 } }
1190       }
1191     }

```

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.

```

1192     \int_compare:nNnT \c@iRow > { -1 }
1193     {
1194       \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@`.

```

1195         { \hrule height \arrayrulewidth width \c_zero_dim }

```

```

1196         }
1197     }
1198 }
1199 }
1200 }

```

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

```

1201 \cs_set_protected:Npn \@@_newcolumnntype #1
1202 {
1203     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1204     \peek_meaning:NTF [
1205         { \newcol@ #1 }
1206         { \newcol@ #1 [ 0 ] }
1207     }

```

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

```

1208 \cs_set_protected:Npn \@@_renew_dots:
1209 {
1210     \cs_set_eq:NN \ldots \@@_Ldots
1211     \cs_set_eq:NN \cdots \@@_Cdots
1212     \cs_set_eq:NN \vdots \@@_Vdots
1213     \cs_set_eq:NN \ddots \@@_Ddots
1214     \cs_set_eq:NN \iddots \@@_Iddots
1215     \cs_set_eq:NN \dots \@@_Ldots
1216     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1217 }

```

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

```

1218 \cs_new_protected:Npn \@@_colortbl_like:
1219 {
1220     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1221     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1222     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1223 }

```

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

```

1224 \cs_new_protected:Npn \@@_pre_array_ii:
1225 {

```

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 will it in each cell.

```

1226     \xglobal \colorlet { nicematrix } { . }

```

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

```

1227     \int_gzero:N \g_@@_total_X_weight_int
1228     \@@_expand_clist:N \l_@@_hlines_clist
1229     \@@_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.

```

1230 \bool_if:NT \c_@@_booktabs_loaded_bool
1231 { \tl_put_left:Nn \BTnormal \@@_create_row_node_i: } % modified in 6.10a
1232 \box_clear_new:N \l_@@_cell_box
1233 \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}`).

```

1234 \bool_if:NT \l_@@_small_bool
1235 {
1236   \cs_set_nopar:Npn \arraystretch { 0.47 }
1237   \dim_set:Nn \arraycolsep { 1.45 pt }
1238 }

1239 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1240 {
1241   \tl_put_right:Nn \@@_begin_of_row:
1242   {
1243     \pgfsys@markposition
1244     { \@@_env: - row - \int_use:N \c@iRow - base }
1245   }
1246 }

```

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`.

```

1247 \cs_set_nopar:Npn \ialign
1248 {
1249   \bool_if:NTF \l_@@_colortbl_loaded_bool
1250   {
1251     \CT@everycr
1252     {
1253       \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1254       \@@_everycr:
1255     }
1256   }
1257   { \everycr { \@@_everycr: } }
1258   \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`.

```

1259 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1260 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1261 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1262 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1263 \dim_gzero_new:N \g_@@_ht_row_one_dim
1264 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1265 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1266 \dim_gzero_new:N \g_@@_ht_last_row_dim
1267 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }

```

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

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

```

1268 \dim_gzero_new:N \g_@@_dp_last_row_dim
1269 \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`.

```

1270 \cs_set_eq:NN \ialign \@@_old_ialign:
1271 \halign
1272 }

```

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).

```

1273 \cs_set_eq:NN \@@_old_ldots \ldots
1274 \cs_set_eq:NN \@@_old_cdots \cdots
1275 \cs_set_eq:NN \@@_old_vdots \vdots
1276 \cs_set_eq:NN \@@_old_ddots \ddots
1277 \cs_set_eq:NN \@@_old_iddots \iddots
1278 \bool_if:NTF \l_@@_standard_cline_bool
1279 { \cs_set_eq:NN \cline \@@_standard_cline }
1280 { \cs_set_eq:NN \cline \@@_cline }
1281 \cs_set_eq:NN \Ldots \@@_Ldots
1282 \cs_set_eq:NN \Cdots \@@_Cdots
1283 \cs_set_eq:NN \Vdots \@@_Vdots
1284 \cs_set_eq:NN \Ddots \@@_Ddots
1285 \cs_set_eq:NN \Iddots \@@_Iddots
1286 \cs_set_eq:NN \Hline \@@_Hline:
1287 \cs_set_eq:NN \Hspace \@@_Hspace:
1288 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1289 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1290 \cs_set_eq:NN \Block \@@_Block:
1291 \cs_set_eq:NN \rotate \@@_rotate:
1292 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1293 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1294 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1295 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1296 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1297 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1298 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1299 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1300 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1301 \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.

```

1302 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1303 \hook_gput_code:nnn { env / tabular / begin } { . }
1304 { \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).

```

1305 \seq_gclear:N \g_@@_multicolumn_cells_seq
1306 \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).

```

1307 \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`).

```

1308 \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.

```

1309   \int_gzero_new:N \g_@@_col_total_int
1310   \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1311   \@@_renew_NC@rewrite@S:
1312   \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.

```

1313   \tl_gclear_new:N \g_@@_Cdots_lines_tl
1314   \tl_gclear_new:N \g_@@_Ldots_lines_tl
1315   \tl_gclear_new:N \g_@@_Vdots_lines_tl
1316   \tl_gclear_new:N \g_@@_Ddots_lines_tl
1317   \tl_gclear_new:N \g_@@_Iddots_lines_tl
1318   \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1319   \tl_gclear_new:N \g_nicematrix_code_before_tl
1320 }

```

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

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

```

1321 \cs_new_protected:Npn \@@_pre_array:
1322 {
1323   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1324   \int_gzero_new:N \c@iRow
1325   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1326   \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).

```

1327   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1328   {
1329     \bool_set_true:N \l_@@_last_row_without_value_bool
1330     \bool_if:NT \g_@@_aux_found_bool
1331     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1332   }
1333   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1334   {
1335     \bool_if:NT \g_@@_aux_found_bool
1336     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1337   }

```

If there is an 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”.

```

1338   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1339   {
1340     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1341     {
1342       \dim_gset:Nn \g_@@_ht_last_row_dim
1343       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1344       \dim_gset:Nn \g_@@_dp_last_row_dim
1345       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1346     }
1347   }

```

```

1348 \seq_gclear:N \g_@@_cols_vlism_seq
1349 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1350 \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.

```

1351 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1352 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1353 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

The command `\create_row_node:` will create a row-node (and not a row of nodes!). However, at the end of the array we construct a “false row” (for the col-nodes) and it interferes with the construction of the last row-node of the array. We don’t want to create such row-node twice (to avoid warnings or, maybe, errors). That’s why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```

1354 \int_gset:Nn \g_@@_last_row_node_int { -1 }

```

The code in `\@@_pre_array_ii:` is used only here.

```

1355 \@@_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.

```

1356 \box_clear_new:N \l_@@_the_array_box

```

We compute the width of both delimiters. We remind that, when the environment `{NiceArray}` is used, it’s possible to specify the delimiters in the preamble (eg `[ccc]`).

```

1357 \dim_zero_new:N \l_@@_left_delim_dim
1358 \dim_zero_new:N \l_@@_right_delim_dim
1359 \bool_if:NTF \g_@@_NiceArray_bool
1360 {
1361   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1362   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1363 }
1364 {

```

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

```

1365 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1366 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1367 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1368 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1369 }

```

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).

```

1370 \hbox_set:Nw \l_@@_the_array_box
1371 \skip_horizontal:N \l_@@_left_margin_dim
1372 \skip_horizontal:N \l_@@_extra_left_margin_dim
1373 \c_math_toggle_token
1374 \bool_if:NTF \l_@@_light_syntax_bool
1375 { \use:c { @@-light-syntax } }
1376 { \use:c { @@-normal-syntax } }
1377 }

```


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

```

1378 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1379 {
1380     \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1381     \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...

```

1382     \@@_pre_array:
1383 }

```

The `\CodeBefore`

The following command will be executed if the `\CodeBefore` has to be actually executed.

```

1384 \cs_new_protected:Npn \@@_pre_code_before:
1385 {

```

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).

```

1386     \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1387     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1388     \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1389     \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_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.

```

1390     \pgfsys@markposition { \@@_env: - position }
1391     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1392     \pgfpicture
1393     \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1394     \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1395     {
1396         \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1397         \pgfcoordinate { \@@_env: - row - ##1 }
1398         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1399     }

```

Now, the recreation of the `col` nodes.

```

1400     \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1401     {
1402         \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1403         \pgfcoordinate { \@@_env: - col - ##1 }
1404         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1405     }

```

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

```

1406     \@@_create_diag_nodes:

```

Now, the creation of the cell nodes (`i-j`), and, maybe also the “medium nodes” and the “large nodes”.

```

1407     \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1408     \endpgfpicture

```

Now, the recreation of the nodes of the blocks *which have a name*.

```

1409     \@@_create_blocks_nodes:

```

```

1410 \bool_if:NT \c_@@_tikz_loaded_bool
1411 {
1412     \tikzset
1413     {
1414         every-picture / .style =
1415         { overlay , name-prefix = \@@_env: - }
1416     }
1417 }
1418 \cs_set_eq:NN \cellcolor \@@_cellcolor
1419 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1420 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1421 \cs_set_eq:NN \rowcolor \@@_rowcolor
1422 \cs_set_eq:NN \rowcolors \@@_rowcolors
1423 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1424 \cs_set_eq:NN \arraycolor \@@_arraycolor
1425 \cs_set_eq:NN \columncolor \@@_columncolor
1426 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1427 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1428 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1429 }

```

```

1430 \cs_new_protected:Npn \@@_exec_code_before:
1431 {
1432     \seq_gclear_new:N \g_@@_colors_seq
1433     \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1434     \group_begin:

```

We compose the `\CodeBefore` in math mode in order to nullify the spaces put by the user between instructions in the `\CodeBefore`.

```

1435     \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`.

```

1436     \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.

```

1437     \@@_actually_color:
1438     \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1439     \group_end:
1440     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1441     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1442 }

```

```

1443 \keys_define:nn { NiceMatrix / CodeBefore }
1444 {
1445     create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1446     create-cell-nodes .default:n = true ,
1447     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1448     sub-matrix .value_required:n = true ,
1449     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1450     delimiters / color .value_required:n = true ,
1451     unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1452 }
1453 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1454 {
1455     \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1456     \@@_CodeBefore:w
1457 }

```

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.

```

1458 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1459 {
1460   \bool_if:NT \g_@@_aux_found_bool
1461   {
1462     \@@_pre_code_before:
1463     #1
1464   }
1465 }

```

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.

```

1466 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1467 {
1468   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1469   {
1470     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1471     \pgfcoordinate { \@@_env: - row - ##1 - base }
1472     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1473     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1474     {
1475       \cs_if_exist:cT
1476       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1477       {
1478         \pgfsys@getposition
1479         { \@@_env: - ##1 - #####1 - NW }
1480         \@@_node_position:
1481         \pgfsys@getposition
1482         { \@@_env: - ##1 - #####1 - SE }
1483         \@@_node_position_i:
1484         \@@_pgf_rect_node:nnn
1485         { \@@_env: - ##1 - #####1 }
1486         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1487         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1488       }
1489     }
1490   }
1491   \int_step_inline:nn \c@iRow
1492   {
1493     \pgfnodealias
1494     { \@@_env: - ##1 - last }
1495     { \@@_env: - ##1 - \int_use:N \c@jCol }
1496   }
1497   \int_step_inline:nn \c@jCol
1498   {
1499     \pgfnodealias
1500     { \@@_env: - last - ##1 }
1501     { \@@_env: - \int_use:N \c@iRow - ##1 }
1502   }
1503   \@@_create_extra_nodes:
1504 }

1505 \cs_new_protected:Npn \@@_create_blocks_nodes:
1506 {
1507   \pgfpicture
1508   \pgf@relevantforpicturesizefalse
1509   \pgfrememberpicturepositiononpagetrue

```

```

1510 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1511 { \@@_create_one_block_node:nnnnn #1 }
1512 \endpgfpicture
1513 }

```

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

```

1514 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1515 {
1516   \tl_if_empty:nF { #5 }
1517   {
1518     \@@_qpoint:n { col - #2 }
1519     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1520     \@@_qpoint:n { #1 }
1521     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1522     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1523     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1524     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1525     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1526     \@@_pgf_rect_node:nnnnn
1527     { \@@_env: - #5 }
1528     { \dim_use:N \l_tmpa_dim }
1529     { \dim_use:N \l_tmpb_dim }
1530     { \dim_use:N \l_@@_tmpc_dim }
1531     { \dim_use:N \l_@@_tmpd_dim }
1532   }
1533 }

1534 \cs_new_protected:Npn \@@_patch_for_revtext:
1535 {
1536   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1537   \cs_set_eq:NN \insert@column \insert@column@array
1538   \cs_set_eq:NN \@classx \@classx@array
1539   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1540   \cs_set_eq:NN \@arraycr \@arraycr@array
1541   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1542   \cs_set_eq:NN \array \array@array
1543   \cs_set_eq:NN \@array \@array@array
1544   \cs_set_eq:NN \@tabular \@tabular@array
1545   \cs_set_eq:NN \@mkpream \@mkpream@array
1546   \cs_set_eq:NN \endarray \endarray@array
1547   \cs_set:Npn \@tabarray { \@ifnextchar [ { \array } { \array [ c ] } }
1548   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1549 }

```

The environment {NiceArrayWithDelims}

```

1550 \NewDocumentEnvironment { NiceArrayWithDelims }
1551 { m m O { } m ! O { } t \CodeBefore }
1552 {
1553   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1554   \@@_provide_pgfsyspdfmark:
1555   \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.

```

1556 \bgroup

```

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).

```

1557 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1558 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1559 \tl_gset:Nn \g_@@_preamble_tl { #4 }

1560 \int_gzero:N \g_@@_block_box_int
1561 \dim_zero:N \g_@@_width_last_col_dim
1562 \dim_zero:N \g_@@_width_first_col_dim
1563 \bool_gset_false:N \g_@@_row_of_col_done_bool
1564 \str_if_empty:NT \g_@@_name_env_str
1565 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1566 \bool_if:NTF \l_@@_NiceTabular_bool
1567 \mode_leave_vertical:
1568 \@@_test_if_math_mode:
1569 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet-in-env } }
1570 \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.

```

1571 \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.

```

1572 \cs_if_exist:NT \tikz@library@external@loaded
1573 {
1574   \tikzexternaldisable
1575   \cs_if_exist:NT \ifstandalone
1576   { \tikzset { external / optimize = false } }
1577 }

```

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

```

1578 \int_gincr:N \g_@@_env_int
1579 \bool_if:NF \l_@@_block_auto_columns_width_bool
1580 { \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`).

```

1581 \seq_gclear:N \g_@@_blocks_seq
1582 \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`.

```

1583 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1584 \seq_gclear:N \g_@@_pos_of_xdots_seq
1585 \tl_gclear_new:N \g_@@_code_before_tl
1586 \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.

```

1587 \bool_gset_false:N \g_@@_aux_found_bool
1588 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1589 {
1590   \bool_gset_true:N \g_@@_aux_found_bool
1591   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1592 }

```

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.

⁶⁶e.g. `\color[rgb]{0.5,0.5,0}`

```

1593 \tl_gclear:N \g_@@_aux_tl
1594 \tl_if_empty:NF \g_@@_code_before_tl
1595 {
1596   \bool_set_true:N \l_@@_code_before_bool
1597   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1598 }

```

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`.

```

1599 \bool_if:NTF \g_@@_NiceArray_bool
1600 { \keys_set:nn { NiceMatrix / NiceArray } }
1601 { \keys_set:nn { NiceMatrix / pNiceArray } }
1602 { #3 , #5 }

```

```

1603 \@@_set_CT@arc@:V \l_@@_rules_color_tl

```

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 `\@@_CodeBefore_Body:w`. After that job, the command `\@@_CodeBefore_Body:w` will go on with `\@@_pre_array:`.

```

1604 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1605 }

```

Now, the second part of the environment `{NiceArrayWithDelims}`.

```

1606 {
1607   \bool_if:NTF \l_@@_light_syntax_bool
1608   { \use:c { end @@-light-syntax } }
1609   { \use:c { end @@-normal-syntax } }
1610   \c_math_toggle_token
1611   \skip_horizontal:N \l_@@_right_margin_dim
1612   \skip_horizontal:N \l_@@_extra_right_margin_dim
1613   \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.

```

1614 \bool_if:NT \l_@@_width_used_bool
1615 {
1616   \int_compare:nNnT \g_@@_total_X_weight_int = 0
1617   { \@@_error:n { width~without~X~columns } }
1618 }

```

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 .

```

1619 \int_compare:nNnT \g_@@_total_X_weight_int > 0
1620 {
1621   \tl_gput_right:Nx \g_@@_aux_tl
1622   {
1623     \bool_set_true:N \l_@@_X_columns_aux_bool
1624     \dim_set:Nn \l_@@_X_columns_dim
1625     {
1626       \dim_compare:nNnTF
1627       {
1628         \dim_abs:n
1629         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1630       }
1631       <
1632       { 0.001 pt }
1633       { \dim_use:N \l_@@_X_columns_dim }

```

```

1634         {
1635             \dim_eval:n
1636             {
1637                 ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1638                 / \int_use:N \g_@@_total_X_weight_int
1639                 + \l_@@_X_columns_dim
1640             }
1641         }
1642     }
1643 }
1644 }

```

It the user has used the key `last-row` with a value, we control that the given value is correct (since we have just constructed the array, we know the real number of rows of the array).

```

1645     \int_compare:nNnT \l_@@_last_row_int > { -2 }
1646     {
1647         \bool_if:NF \l_@@_last_row_without_value_bool
1648         {
1649             \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1650             {
1651                 \@@_error:n { Wrong~last~row }
1652                 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1653             }
1654         }
1655     }

```

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

```

1656     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1657     \bool_if:nTF \g_@@_last_col_found_bool
1658     { \int_gdecr:N \c@jCol }
1659     {
1660         \int_compare:nNnT \l_@@_last_col_int > { -1 }
1661         { \@@_error:n { last~col~not~used } }
1662     }

```

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

```

1663     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1664     \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. 131).

```

1665     \int_compare:nNnT \l_@@_first_col_int = 0
1666     {
1667         \skip_horizontal:N \col@sep
1668         \skip_horizontal:N \g_@@_width_first_col_dim
1669     }

```

The construction of the real box is different when `\g_@@_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.

```

1670     \bool_if:NTF \g_@@_NiceArray_bool
1671     {
1672         \str_case:VnF \l_@@_baseline_tl
1673         {
1674             b \@@_use_arraybox_with_notes_b:
1675             c \@@_use_arraybox_with_notes_c:
1676         }
1677         \@@_use_arraybox_with_notes:

```

⁶⁷We remind that the potential “first column” (exterior) has the number 0.

1678 }

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).

```
1679 {
1680   \int_compare:nNnTF \l_@@_first_row_int = 0
1681   {
1682     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1683     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1684   }
1685   { \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

```
1686   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1687   {
1688     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1689     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1690   }
1691   { \dim_zero:N \l_tmpb_dim }
1692   \hbox_set:Nn \l_tmpa_box
1693   {
1694     \c_math_toggle_token
1695     \@@_color:V \l_@@_delimiters_color_tl
1696     \exp_after:wN \left \g_@@_left_delim_tl
1697     \vcenter
1698     {
```

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.

```
1699     \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1700     \hbox
1701     {
1702       \bool_if:NTF \l_@@_NiceTabular_bool
1703       { \skip_horizontal:N -\tabcolsep }
1704       { \skip_horizontal:N -\arraycolsep }
1705       \@@_use_arraybox_with_notes_c:
1706       \bool_if:NTF \l_@@_NiceTabular_bool
1707       { \skip_horizontal:N -\tabcolsep }
1708       { \skip_horizontal:N -\arraycolsep }
1709     }
```

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

```
1710     \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1711   }
```

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.

```
1712     \@@_color:V \l_@@_delimiters_color_tl
1713     \exp_after:wN \right \g_@@_right_delim_tl
1714     \c_math_toggle_token
1715   }
```

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.

```
1716   \bool_if:NTF \l_@@_delimiters_max_width_bool
1717   {
1718     \@@_put_box_in_flow_bis:nn
1719     \g_@@_left_delim_tl \g_@@_right_delim_tl
1720   }
1721   \@@_put_box_in_flow:
```

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).


```

1722     }
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. 132).
1723     \bool_if:NT \g_@@_last_col_found_bool
1724     {
1725         \skip_horizontal:N \g_@@_width_last_col_dim
1726         \skip_horizontal:N \col@sep
1727     }
1728     \bool_if:NF \l_@@_Matrix_bool
1729     {
1730         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1731         {
1732             \@@_error:n { columns-not-used }
1733             \group_begin:
1734             \globaldefs = 1
1735             \@@_msg_redirect_name:nn { columns-not-used } { none }
1736             \group_end:
1737         }
1738     }
1739     \@@_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.

```

1740     \egroup

```

We want to write on the aux file all the informations corresponding to the current environment.

```

1741     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1742     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1743     \iow_now:Nx \@mainaux
1744     {
1745         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _tl }
1746         { \exp_not:V \g_@@_aux_tl }
1747     }
1748     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1749     \bool_if:NT \c_@@_footnote_bool \endsavenotes
1750 }

```

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.⁶⁹

The preamble given by the final user is in `\g_@@_preamble_tl` and the modified version will be stored in `\g_@@_preamble_tl` also.

```

1751 \cs_new_protected:Npn \@@_transform_preamble:
1752 {

```

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.

⁶⁹Be careful: the transformation of the preamble may also have by-side effects, for example, the boolean `\g_@@_NiceArray_bool` will be set to `false` if we detect in the preamble a delimiter at the beginning or at the end.

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 to use 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.

```
1753 \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).

```
1754 \bool_if:NF \l_@@_Matrix_bool
1755 {
1756   \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1757   \@@_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).

```
1758 \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).

```
1759 \exp_args:NV \@temptokena \g_@@_preamble_tl
```

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

```
1760 \@tempswatrue
```

The following line actually does the expansion (it's has been copied from `array.sty`). The expanded version is still in `\@temptokena`.

```
1761 \@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`.

```
1762 \int_gzero:N \c@jCol
1763 \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1764 \bool_gset_false:N \g_tmpb_bool
1765 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1766 {
1767   \tl_gset:Nn \g_@@_preamble_tl
1768   { ! { \skip_horizontal:N \arrayrulewidth } }
1769 }
1770 {
1771   \clist_if_in:NnT \l_@@_vlines_clist 1
1772   {
1773     \tl_gset:Nn \g_@@_preamble_tl
1774     { ! { \skip_horizontal:N \arrayrulewidth } }
1775   }
1776 }
```

The sequence `\g_@@_cols_vlism_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 `vlism`).

```
1777 \seq_clear:N \g_@@_cols_vlism_seq
```

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

```
1778 \int_zero:N \l_tmpa_int
```

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

```
1779 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1780 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1781 }
```

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

```

1782   \bool_if:NT \l_@@_colortbl_like_bool
1783   {
1784     \regex_replace_all:NnN
1785       \c_@@_columncolor_regex
1786       { \c { @@_columncolor_preamble } }
1787       \g_@@_preamble_tl
1788   }

```

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

```

1789   \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}`.

```

1790   \bool_lazy_or:nnT
1791     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1792     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1793     { \bool_gset_false:N \g_@@_NiceArray_bool }

```

We want to remind whether there is a specifier `|` at the end of the preamble.

```

1794   \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).

```

1795   \int_compare:nNnTF \l_@@_first_col_int = 0
1796   { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1797   {
1798     \bool_lazy_all:nT
1799     {
1800       \g_@@_NiceArray_bool
1801       { \bool_not_p:n \l_@@_NiceTabular_bool }
1802       { \tl_if_empty_p:N \l_@@_vlines_clist }
1803       { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1804     }
1805     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1806   }
1807   \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1808   { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1809   {
1810     \bool_lazy_all:nT
1811     {
1812       \g_@@_NiceArray_bool
1813       { \bool_not_p:n \l_@@_NiceTabular_bool }
1814       { \tl_if_empty_p:N \l_@@_vlines_clist }
1815       { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1816     }
1817     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1818   }

```

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`).

```

1819   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1820   {
1821     \tl_gput_right:Nn \g_@@_preamble_tl
1822       { > { \@@_error_too_much_cols: } 1 }
1823   }
1824 }

```

The command `\@@_patch_preamble:n` is the main function for the transformation of the preamble. It is recursive.

```

1825 \cs_new_protected:Npn \@@_patch_preamble:n #1

```

```

1826 {
1827   \str_case:nnF { #1 }
1828   {
1829     c { \@@_patch_preamble_i:n #1 }
1830     l { \@@_patch_preamble_i:n #1 }
1831     r { \@@_patch_preamble_i:n #1 }
1832     > { \@@_patch_preamble_ii:nn #1 }
1833     ! { \@@_patch_preamble_ii:nn #1 }
1834     @ { \@@_patch_preamble_ii:nn #1 }
1835     | { \@@_patch_preamble_iii:n #1 }
1836     p { \@@_patch_preamble_iv:n #1 }
1837     b { \@@_patch_preamble_iv:n #1 }
1838     m { \@@_patch_preamble_iv:n #1 }
1839     \@@_V: { \@@_patch_preamble_v:n }
1840     V { \@@_patch_preamble_v:n }
1841     \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
1842     \@@_W: { \@@_patch_preamble_vi:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1843     \@@_S: { \@@_patch_preamble_vii:n }
1844     ( { \@@_patch_preamble_viii:nn #1 }
1845     [ { \@@_patch_preamble_viii:nn #1 }
1846     \{ { \@@_patch_preamble_viii:nn #1 }
1847     ) { \@@_patch_preamble_ix:nn #1 }
1848     ] { \@@_patch_preamble_ix:nn #1 }
1849     \} { \@@_patch_preamble_ix:nn #1 }
1850     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}`.

```

1851   \@@_X { \@@_patch_preamble_x:n }
1852   \q_stop { }
1853 }
1854 {
1855   \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
1856   {
1857     \seq_gput_right:Nx \g_@@_cols_vlism_seq
1858     { \int_eval:n { \c@jCol + 1 } }
1859     \tl_gput_right:Nx \g_@@_preamble_tl
1860     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1861     \@@_patch_preamble:n
1862   }

```

Now the case of a letter set by the final user for a customized rule. Such customized rule is defined by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs. Among the keys available in that list, there is the key `letter`. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix/ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```

1863   {
1864     \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
1865     {
1866       \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
1867       \@@_patch_preamble:n
1868     }
1869     { \@@_fatal:nn { unknown~column~type } { #1 } }
1870   }
1871 }
1872 }

```

Now, we will list all the auxiliary functions for the different types of entries in the preamble of the array.

For `c`, `l` and `r`

```

1873 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1874 {
1875   \tl_gput_right:Nn \g_@@_preamble_tl
1876   {
1877     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
1878     #1
1879     < \@@_cell_end:
1880   }

```

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

```

1881   \int_gincr:N \c@jCol
1882   \@@_patch_preamble_xi:n
1883 }

```

For >, ! and @

```

1884 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1885 {
1886   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1887   \@@_patch_preamble:n
1888 }

```

For |

```

1889 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1890 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1891   \int_incr:N \l_tmpa_int
1892   \@@_patch_preamble_iii_i:n
1893 }

```

```

1894 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1895 {
1896   \str_if_eq:nnTF { #1 } |
1897   { \@@_patch_preamble_iii:n | }
1898   {
1899     \tl_gput_right:Nx \g_@@_preamble_tl
1900     {
1901       \exp_not:N !
1902       {
1903         \skip_horizontal:n
1904         {

```

Here, the command \dim_eval:n is mandatory.

```

1905         \dim_eval:n
1906         {
1907           \arrayrulewidth * \l_tmpa_int
1908           + \doublerulesep * ( \l_tmpa_int - 1 )
1909         }
1910       }
1911     }
1912   }
1913   \tl_gput_right:Nx \g_@@_internal_code_after_tl
1914   {
1915     \@@_vline:n
1916     {
1917       position = \int_eval:n { \c@jCol + 1 } ,
1918       multiplicity = \int_use:N \l_tmpa_int ,
1919     }

```

We don't have provided value for start nor for end, which means that the rule will cover (potentially) all the rows of the array.

```

1920   }
1921   \int_zero:N \l_tmpa_int
1922   \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
1923   \@@_patch_preamble:n #1

```

```

1924     }
1925   }
1926 \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.

```

1927 \keys_define:nn { WithArrows / p-column }
1928 {
1929   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
1930   r .value_forbidden:n = true ,
1931   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
1932   c .value_forbidden:n = true ,
1933   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
1934   l .value_forbidden:n = true ,
1935   si .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
1936   si .value_forbidden:n = true ,
1937   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
1938   p .value_forbidden:n = true ,
1939   t .meta:n = p ,
1940   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
1941   m .value_forbidden:n = true ,
1942   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
1943   b .value_forbidden:n = true ,
1944 }

```

For `p`, `b` and `m`. The argument `#1` is that value : `p`, `b` or `m`.

```

1945 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
1946 {
1947   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

Now, you look for a potential character [after the letter of the specifier (for the options).

```

1948   \@@_patch_preamble_iv_i:n
1949 }

1950 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
1951 {
1952   \str_if_eq:nnTF { #1 } { [ ]
1953     { \@@_patch_preamble_iv_ii:w [ ]
1954       { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
1955     }

1956   \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
1957     { \@@_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.

```

1958 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
1959 {

```

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).

```

1960   \str_set:Nn \l_@@_hpos_col_str { j }
1961   \keys_set:nn { WithArrows / p-column } { #1 }
1962   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
1963 }

```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`.

```

1964 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
1965 {
1966   \use:x
1967   {

```

```

1968 \@@_patch_preamble_iv_v:nnnnnnnn
1969 { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
1970 { \dim_eval:n { #1 } }
1971 {

```

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.

```

1972 \str_if_eq:VnTF \l_@@_hpos_col_str j
1973 { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
1974 {
1975 \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
1976 { \l_@@_hpos_col_str }
1977 }
1978 \str_case:Vn \l_@@_hpos_col_str
1979 {
1980 c { \exp_not:N \centering }
1981 l { \exp_not:N \raggedright }
1982 r { \exp_not:N \raggedleft }
1983 }
1984 }
1985 { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
1986 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
1987 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
1988 { #2 }
1989 {
1990 \str_case:VnF \l_@@_hpos_col_str
1991 {
1992 { j } { c }
1993 { si } { c }
1994 }
1995 { \l_@@_hpos_col_str }
1996 }
1997 }

```

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

```

1998 \int_gincr:N \c@jCol
1999 \@@_patch_preamble_xi:n
2000 }

```

#1 is the optional argument of `{minipage}` (or `{varwidth}`): `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}` (or `{varwidth}`), 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` (or `r` or `l`: see **#8**).

#6 is a code put just after the `c` (or `r` or `l`: see **#8**).

#7 is the type of environment: `minipage` or `varwidth`.

#8 is the lettre `c` or `r` or `l` which is the basic specifier of column which is used *in fine*.

```

2001 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2002 {
2003 \tl_gput_right:Nn \g_@@_preamble_tl
2004 {
2005 > {

```

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.

```

2006 \dim_set:Nn \l_@@_col_width_dim { #2 }

```

```

2007         \@@_cell_begin:w
2008         \begin { #7 } [ #1 ] { #2 }

```

The following lines have been taken from `array.sty`.

```

2009         \everypar
2010         {
2011             \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2012             \everypar { }
2013         }

```

Now, the potential code for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing).

```

2014         #3

```

The following code is to allow something like `\centering` in `\RowStyle`.

```

2015         \g_@@_row_style_tl
2016         \arraybackslash
2017         #5
2018     }
2019     #8
2020     < {
2021         #6

```

The following line has been taken from `array.sty`.

```

2022         \@finalstrut \@arstrutbox
2023         % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
2024         \end { #7 }

```

If the letter in the preamble is `m`, `#4` will be equal to `\@@_center_cell_box:` (see just below).

```

2025         #4
2026         \@@_cell_end:
2027     }
2028 }
2029 }

```

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.

```

2030 \cs_new_protected:Npn \@@_center_cell_box:
2031 {

```

By putting instructions in `\g_@@_post_action_cell_tl`, we require a post-action of the box `\l_@@_cell_box`.

```

2032     \tl_gput_right:Nn \g_@@_post_action_cell_tl
2033     {
2034         \int_compare:nNnT
2035         { \box_ht:N \l_@@_cell_box }
2036         >
2037         { \box_ht:N \@arstrutbox }
2038         {
2039             \hbox_set:Nn \l_@@_cell_box
2040             {
2041                 \box_move_down:nn
2042                 {
2043                     ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2044                     + \baselineskip ) / 2
2045                 }
2046                 { \box_use:N \l_@@_cell_box }
2047             }
2048         }
2049     }
2050 }

```


For V (similar to the V of varwidth).

```

2051 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2052 {
2053   \str_if_eq:nnTF { #1 } { [ ]
2054     { \@@_patch_preamble_v_i:w [ ]
2055       { \@@_patch_preamble_v_i:w [ ] { #1 } }
2056     }
2057   \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2058     { \@@_patch_preamble_v_ii:nn { #1 } }
2059   \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2060     {
2061       \str_set:Nn \l_@@_vpos_col_str { p }
2062       \str_set:Nn \l_@@_hpos_col_str { j }
2063       \keys_set:nn { WithArrows / p-column } { #1 }
2064       \bool_if:NTF \c_@@_varwidth_loaded_bool
2065         { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2066         {
2067           \@@_error:n { varwidth~not~loaded }
2068           \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2069         }
2070     }

```

For w and W

```

2071 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2072 {
2073   \tl_gput_right:Nn \g_@@_preamble_tl
2074   {
2075     > {

```

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.

```

2076       \dim_set:Nn \l_@@_col_width_dim { #4 }
2077       \hbox_set:Nw \l_@@_cell_box
2078       \@@_cell_begin:w
2079       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2080     }
2081   c
2082   < {
2083     \@@_cell_end:
2084     #1
2085     \hbox_set_end:
2086     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2087     \@@_adjust_size_box:
2088     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2089   }
2090 }

```

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

```

2091   \int_gincr:N \c@jCol
2092   \@@_patch_preamble_xi:n
2093 }

```

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).

```

2094 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2095 {
2096   \str_if_eq:nnTF { #1 } { [ ]
2097     { \@@_patch_preamble_vii_i:w [ ]
2098       { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2099     }
2100   \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2101     { \@@_patch_preamble_vii_ii:n { #1 } }

```

```

2102 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2103 {

```

We test whether the version of nicematrix is at least 3.0. We will change the programming of the test further with something like `\@ifpackagelater`.

```

2104 \cs_if_exist:NTF \siunitx_cell_begin:w
2105 {
2106   \tl_gput_right:Nn \g_@@_preamble_tl
2107   {
2108     > {
2109       \@@_cell_begin:w
2110       \keys_set:nn { siunitx } { #1 }
2111       \siunitx_cell_begin:w
2112     }
2113     c
2114     < { \siunitx_cell_end: \@@_cell_end: }
2115   }

```

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

```

2116 \int_gincr:N \c@jCol
2117 \@@_patch_preamble_xi:n
2118 }
2119 { \@@_fatal:n { Version-of-siunitx-too-old } }
2120 }

```

For `(`, `[` and `\{`.

```

2121 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2122 {
2123   \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.

```

2124 \int_compare:nNnTF \c@jCol = \c_zero_int
2125 {
2126   \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2127   {

```

In that case, in fact, the first letter of the preamble must be considered as the left delimiter of the array.

```

2128 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2129 \tl_gset:Nn \g_@@_right_delim_tl { . }
2130 \@@_patch_preamble:n #2
2131 }
2132 {
2133   \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2134   \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2135 }
2136 }
2137 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2138 }
2139 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2140 {
2141   \tl_gput_right:Nx \g_@@_internal_code_after_tl
2142   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2143   \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
2144   {
2145     \@@_error:nn { delimiter~after~opening } { #2 }
2146     \@@_patch_preamble:n
2147   }
2148   { \@@_patch_preamble:n #2 }
2149 }

```

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}).

```

2150 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2151 {
2152   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter-with-small } }
2153   \tl_if_in:nnTF { ) ] \} } { #2 }
2154   { \@@_patch_preamble_ix_i:nnn #1 #2 }
2155   {
2156     \tl_if_eq:nnTF { \q_stop } { #2 }
2157     {
2158       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2159       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2160       {
2161         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2162         \tl_gput_right:Nx \g_@@_internal_code_after_tl
2163         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2164         \@@_patch_preamble:n #2
2165       }
2166     }
2167     {
2168       \tl_if_in:nnT { ( [ \{ } { #2 }
2169       { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2170       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2171       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2172       \@@_patch_preamble:n #2
2173     }
2174   }
2175 }

2176 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2177 {
2178   \tl_if_eq:nnTF { \q_stop } { #3 }
2179   {
2180     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2181     {
2182       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2183       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2184       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2185       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2186     }
2187     {
2188       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2189       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2190       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2191       \@@_error:nn { double-closing-delimiter } { #2 }
2192     }
2193   }
2194   {
2195     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2196     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2197     \@@_error:nn { double-closing-delimiter } { #2 }
2198     \@@_patch_preamble:n #3
2199   }
2200 }

```

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.

```

2201 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2202 {
2203   \str_if_eq:nnTF { #1 } { [ ]
2204   { \@@_patch_preamble_x_i:w [ ]
2205   { \@@_patch_preamble_x_i:w [ ] #1 }
2206 }

```

```

2207 \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2208 { \@@_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).

```

2209 \keys_define:nn { WithArrows / X-column }
2210 { 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.

```

2211 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2212 {

```

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).

```

2213 \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).

```

2214 \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.

```

2215 \int_zero_new:N \l_@@_weight_int
2216 \int_set:Nn \l_@@_weight_int { 1 }
2217 \keys_set:known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
2218 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2219 \int_compare:nNnT \l_@@_weight_int < 0
2220 {
2221 \@@_error:nx { negative-weight } { \int_use:N \l_@@_weight_int }
2222 \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2223 }
2224 \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).

```

2225 \bool_if:NTF \l_@@_X_columns_aux_bool
2226 {
2227 \@@_patch_preamble_iv_iv:nn
2228 { \l_@@_weight_int \l_@@_X_columns_dim }
2229 { minipage }
2230 }
2231 {
2232 \tl_gput_right:Nn \g_@@_preamble_tl
2233 {
2234 > {
2235 \@@_cell_begin:w
2236 \bool_set_true:N \l_@@_X_column_bool

```

The following code will nullify the box of the cell.

```

2237 \tl_gput_right:Nn \g_@@_post_action_cell_tl
2238 { \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.

```

2239 \begin { minipage } { 5 cm } \arraybackslash
2240 }
2241 c
2242 < {
2243 \end { minipage }
2244 \@@_cell_end:
2245 }

```

```

2246     }
2247     \int_gincr:N \c@jCol
2248     \@@_patch_preamble_xi:n
2249   }
2250 }

```

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.

```

2251 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2252 {
2253   \str_if_eq:nnTF { #1 } { < }
2254   \@@_patch_preamble_xiii:n
2255   {
2256     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2257     {
2258       \tl_gput_right:Nn \g_@@_preamble_tl
2259       { ! { \skip_horizontal:N \arrayrulewidth } }
2260     }
2261     {
2262       \exp_args:NNx
2263       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2264       {
2265         \tl_gput_right:Nn \g_@@_preamble_tl
2266         { ! { \skip_horizontal:N \arrayrulewidth } }
2267       }
2268     }
2269     \@@_patch_preamble:n { #1 }
2270   }
2271 }
2272 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2273 {
2274   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2275   \@@_patch_preamble_xi:n
2276 }
2277 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2278 {
2279   \@temptokena { #2 }
2280   \@tempswatrue
2281   \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2282   \tl_gclear:N \g_@@_preamble_tl
2283   \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
2284   \tl_set_eq:NN #1 \g_@@_preamble_tl
2285 }

```

The redefinition of `\multicolumn`

The following command must *not* be protected since it begins with `\multispan` (a TeX primitive).

```

2286 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2287 {

```

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`.

```

2288   \multispan { #1 }
2289   \begingroup
2290   \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

You do the expansion of the (small) preamble with the tools of array.

```

2291 \temptokena = { #2 }
2292 \tempswatrue
2293 \@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.

```

2294 \tl_gclear:N \g_@@_preamble_tl
2295 \exp_after:wN \@@_patch_m_preamble:n \the \temptokena \q_stop

```

The following lines are an adaptation of the definition of `\multicolumn` in array.

```

2296 \exp_args:NV \mkpream \g_@@_preamble_tl
2297 \addtopreamble \empty
2298 \endgroup

```

Now, you do a treatment specific to nicematrix which has no equivalent in the original definition of `\multicolumn`.

```

2299 \int_compare:nNnT { #1 } > 1
2300 {
2301   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2302   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2303   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2304   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2305   {
2306     {
2307       \int_compare:nNnTF \c@jCol = 0
2308       { \int_eval:n { \c@iRow + 1 } }
2309       { \int_use:N \c@iRow }
2310     }
2311     { \int_eval:n { \c@jCol + 1 } }
2312     {
2313       \int_compare:nNnTF \c@jCol = 0
2314       { \int_eval:n { \c@iRow + 1 } }
2315       { \int_use:N \c@iRow }
2316     }
2317     { \int_eval:n { \c@jCol + #1 } }
2318     { } % for the name of the block
2319   }
2320 }

```

The following lines were in the original definition of `\multicolumn`.

```

2321 \cs_set:Npn \@sharp { #3 }
2322 \@arstrut
2323 \@preamble
2324 \null

```

We add some lines.

```

2325 \int_gadd:Nn \c@jCol { #1 - 1 }
2326 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2327 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2328 \ignorespaces
2329 }

```

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`.

```

2330 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2331 {
2332   \str_case:nnF { #1 }
2333   {
2334     c { \@@_patch_m_preamble_i:n #1 }
2335     l { \@@_patch_m_preamble_i:n #1 }
2336     r { \@@_patch_m_preamble_i:n #1 }
2337     > { \@@_patch_m_preamble_ii:nn #1 }

```

```

2338 ! { \@@_patch_m_preamble_ii:nn #1 }
2339 @ { \@@_patch_m_preamble_ii:nn #1 }
2340 | { \@@_patch_m_preamble_iii:n #1 }
2341 p { \@@_patch_m_preamble_iv:nnn t #1 }
2342 m { \@@_patch_m_preamble_iv:nnn c #1 }
2343 b { \@@_patch_m_preamble_iv:nnn b #1 }
2344 \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2345 \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2346 \q_stop { }
2347 }
2348 { \@@_fatal:nn { unknown~column~type } { #1 } }
2349 }

```

For c, l and r

```

2350 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2351 {
2352   \tl_gput_right:Nn \g_@@_preamble_tl
2353   {
2354     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2355     #1
2356     < \@@_cell_end:
2357   }

```

We test for the presence of a <.

```

2358   \@@_patch_m_preamble_x:n
2359 }

```

For >, ! and @

```

2360 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2361 {
2362   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2363   \@@_patch_m_preamble:n
2364 }

```

For |

```

2365 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2366 {
2367   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2368   \@@_patch_m_preamble:n
2369 }

```

For p, m and b

```

2370 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2371 {
2372   \tl_gput_right:Nn \g_@@_preamble_tl
2373   {
2374     > {
2375       \@@_cell_begin:w
2376       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2377       \mode_leave_vertical:
2378       \arraybackslash
2379       \vrule height \box_ht:N \@@arstrutbox depth 0 pt width 0 pt
2380     }
2381     c
2382     < {
2383       \vrule height 0 pt depth \box_dp:N \@@arstrutbox width 0 pt
2384       \end { minipage }
2385       \@@_cell_end:
2386     }
2387   }

```

We test for the presence of a <.

```

2388   \@@_patch_m_preamble_x:n
2389 }

```

For w and W

```

2390 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2391 {
2392   \tl_gput_right:Nn \g_@@_preamble_tl
2393   {
2394     > {
2395       \hbox_set:Nw \l_@@_cell_box
2396       \@@_cell_begin:w
2397       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2398     }
2399     c
2400     < {
2401       \@@_cell_end:
2402       #1
2403       \hbox_set_end:
2404       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2405       \@@_adjust_size_box:
2406       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2407     }
2408   }

```

We test for the presence of a <.

```

2409   \@@_patch_m_preamble_x:n
2410 }

```

After a specifier of column, we have to test whether there is one or several <{..}.

```

2411 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2412 {
2413   \str_if_eq:nnTF { #1 } { < }
2414   \@@_patch_m_preamble_ix:n
2415   { \@@_patch_m_preamble:n { #1 } }
2416 }
2417 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2418 {
2419   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2420   \@@_patch_m_preamble_x:n
2421 }

```

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).

```

2422 \cs_new_protected:Npn \@@_put_box_in_flow:
2423 {
2424   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2425   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2426   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2427   { \box_use_drop:N \l_tmpa_box }
2428   \@@_put_box_in_flow_i:
2429 }

```

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).

```

2430 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2431 {
2432   \pgfpicture
2433   \@@_qpoint:n { row - 1 }
2434   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2435   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2436   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2437   \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).

```

2438 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2439 {
2440   \int_set:Nn \l_tmpa_int
2441   {
2442     \str_range:Nnn
2443     \l_@@_baseline_tl
2444     6
2445     { \tl_count:V \l_@@_baseline_tl }
2446   }
2447   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2448 }
2449 {
2450   \str_case:VnF \l_@@_baseline_tl
2451   {
2452     { t } { \int_set:Nn \l_tmpa_int 1 }
2453     { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2454   }
2455   { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2456   \bool_lazy_or:nnT
2457   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2458   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2459   {
2460     \@@_error:n { bad~value~for~baseline }
2461     \int_set:Nn \l_tmpa_int 1
2462   }
2463   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2464 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2465 }
2466 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2467 \endpgfpicture
2468 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2469 \box_use_drop:N \l_tmpa_box
2470 }

```

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).

```

2471 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2472 {

```

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.

```

2473 \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
2474 {
2475   \box_set_wd:Nn \l_@@_the_array_box
2476   { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2477 }

```

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

```

2478 \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.

```

2479 \hbox
2480 {
2481   \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.

```

2482     \@@_create_extra_nodes:
2483     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2484 }

```

We don't do the following test with `\c@tabularnote` because the value of that counter is not reliable when the command `\ttabbox` of `floatrow` is used (because `\ttabbox` de-activate `\stepcounter` because if compiles several times its tabular).

```

2485     \bool_lazy_or:nnT
2486     { ! \seq_if_empty_p:N \g_@@_tabularnotes_seq }
2487     { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2488     \@@_insert_tabularnotes:
2489     \end { minipage }
2490 }

2491 \cs_new_protected:Npn \@@_insert_tabularnotes:
2492 {
2493     \skip_vertical:N 0.65ex

```

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

```

2494     \group_begin:
2495     \l_@@_notes_code_before_tl
2496     \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.

```

2497     \int_compare:nNnT \c@tabularnote > 0
2498     {
2499         \bool_if:NTF \l_@@_notes_para_bool
2500         {
2501             \begin { tabularnotes* }
2502             \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2503             \end { tabularnotes* }

```

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

```

2504         \par
2505     }
2506     {
2507         \tabularnotes
2508         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2509         \endtabularnotes
2510     }
2511 }
2512 \unskip
2513 \group_end:
2514 \bool_if:NT \l_@@_notes_bottomrule_bool
2515 {
2516     \bool_if:NTF \c_@@_booktabs_loaded_bool
2517     {

```

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

```

2518         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2519         { \CT@arc@ \hrule height \heavyrulewidth }
2520     }
2521     { \@@_error:n { bottomrule~without~booktabs } }
2522 }
2523 \l_@@_notes_code_after_tl
2524 \seq_gclear:N \g_@@_tabularnotes_seq
2525 \int_gzero:N \c@tabularnote
2526 }

```

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`.

```

2527 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2528 {
2529   \pgfpicture
2530     \@@_qpoint:n { row - 1 }
2531     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2532     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2533     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2534   \endpgfpicture
2535   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2536   \int_compare:nNnT \l_@@_first_row_int = 0
2537   {
2538     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2539     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2540   }
2541   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2542 }

```

Now, the general case.

```

2543 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2544 {

```

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

```

2545   \tl_if_eq:NnT \l_@@_baseline_tl { t }
2546   { \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`.

```

2547   \pgfpicture
2548   \@@_qpoint:n { row - 1 }
2549   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2550   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2551   {
2552     \int_set:Nn \l_tmpa_int
2553     {
2554       \str_range:Nnn
2555         \l_@@_baseline_tl
2556         6
2557         { \tl_count:V \l_@@_baseline_tl }
2558     }
2559     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2560   }
2561   {
2562     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2563     \bool_lazy_or:nnT
2564       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2565       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2566     {
2567       \@@_error:n { bad-value-for-baseline }
2568       \int_set:Nn \l_tmpa_int 1
2569     }
2570     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2571   }
2572   \dim_gsub:Nn \g_tmpa_dim \pgf@y
2573   \endpgfpicture
2574   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2575   \int_compare:nNnT \l_@@_first_row_int = 0
2576   {
2577     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2578     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2579   }
2580   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

2581 }

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.

```
2582 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2583 {
```

We will compute the real width of both delimiters used.

```
2584 \dim_zero_new:N \l_@@_real_left_delim_dim
2585 \dim_zero_new:N \l_@@_real_right_delim_dim
2586 \hbox_set:Nn \l_tmpb_box
2587 {
2588   \c_math_toggle_token
2589   \left #1
2590   \vcenter
2591   {
2592     \vbox_to_ht:nn
2593     { \box_ht_plus_dp:N \l_tmpa_box }
2594     { }
2595   }
2596   \right .
2597   \c_math_toggle_token
2598 }
2599 \dim_set:Nn \l_@@_real_left_delim_dim
2600 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2601 \hbox_set:Nn \l_tmpb_box
2602 {
2603   \c_math_toggle_token
2604   \left .
2605   \vbox_to_ht:nn
2606   { \box_ht_plus_dp:N \l_tmpa_box }
2607   { }
2608   \right #2
2609   \c_math_toggle_token
2610 }
2611 \dim_set:Nn \l_@@_real_right_delim_dim
2612 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
2613 \skip_horizontal:N \l_@@_left_delim_dim
2614 \skip_horizontal:N -\l_@@_real_left_delim_dim
2615 \@@_put_box_in_flow:
2616 \skip_horizontal:N \l_@@_right_delim_dim
2617 \skip_horizontal:N -\l_@@_real_right_delim_dim
2618 }
```

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).

```
2619 \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).

```
2620 {
2621   \peek_remove_spaces:n
2622   {
2623     \peek_meaning:NTF \end
2624     \@@_analyze_end:Nn
```

```

2625     {
2626         \@@_transform_preamble:
Here is the call to \array (we have a dedicated macro \@@_array:n because of compatibility with
the classes revtex4-1 and revtex4-2).
2627         \@@_array:V \g_@@_preamble_tl
2628     }
2629 }
2630 }
2631 {
2632     \@@_create_col_nodes:
2633     \endarray
2634 }

```

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

```

2635 \NewDocumentEnvironment { @@-light-syntax } { b }
2636 {

```

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`.

```

2637     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2638     \tl_map_inline:nn { #1 }
2639     {
2640         \str_if_eq:nnT { ##1 } { & }
2641         { \@@_fatal:n { ampersand-in-light-syntax } }
2642         \str_if_eq:nnT { ##1 } { \ }
2643         { \@@_fatal:n { double-backslash-in-light-syntax } }
2644     }

```

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`.

```

2645     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

The command `\array` is hidden somewhere in `\@@_light_syntax_i:w`.

```

2646 }

```

Now, the second part of the environment. We must leave these lines in the second part (and not put them in the first part even though we caught the whole body of the environment with an argument of type `b`) in order to have the columns `S` of `siunitx` working fine.

```

2647 {
2648     \@@_create_col_nodes:
2649     \endarray
2650 }
2651 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
2652 {
2653     \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).

```

2654     \seq_clear_new:N \l_@@_rows_seq

```

We rescan the character of end of line in order to have the correct catcode.

```

2655     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2656     \seq_set_split:NvN \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

2657     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
2658     \tl_if_empty:NF \l_tmpa_tl
2659     { \seq_put_right:Nv \l_@@_rows_seq \l_tmpa_tl }

```

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 compute 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.

```
2660 \int_compare:nNnT \l_@@_last_row_int = { -1 }
2661 { \int_set:Nn \l_@@_last_row_int { \seq_count:N \l_@@_rows_seq } }
```

The new value of the body (that is to say after replacement of the separators of rows and columns by `\\` and `&`) of the environment will be stored in `\l_@@_new_body_tl` (that part of the implementation has been changed in the version 6.11 of `nicematrix` in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`).

```
2662 \tl_clear_new:N \l_@@_new_body_tl
2663 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
2664 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
2665 \@@_line_with_light_syntax:V \l_tmpa_tl
```

Now, the other rows (with the same treatment, excepted that we have to insert `\\` between the rows).

```
2666 \seq_map_inline:Nn \l_@@_rows_seq
2667 {
2668   \tl_put_right:Nn \l_@@_new_body_tl { \\ }
2669   \@@_line_with_light_syntax:n { ##1 }
2670 }
2671 \int_compare:nNnT \l_@@_last_col_int = { -1 }
2672 {
2673   \int_set:Nn \l_@@_last_col_int
2674   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
2675 }
```

Now, we can construct the preamble: if the user has used the key `last-col`, we have the correct number of columns even though the user has used `last-col` without value.

```
2676 \@@_transform_preamble:
```

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

```
2677 \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
2678 }
2679 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2680 {
2681   \seq_clear_new:N \l_@@_cells_seq
2682   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
2683   \int_set:Nn \l_@@_nb_cols_int
2684   {
2685     \int_max:nn
2686     \l_@@_nb_cols_int
2687     { \seq_count:N \l_@@_cells_seq }
2688   }
2689   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
2690   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
2691   \seq_map_inline:Nn \l_@@_cells_seq
2692   { \tl_put_right:Nn \l_@@_new_body_tl { & ##1 } }
2693 }
2694 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { V }
```

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). When this command is used, `#1` is, in fact, always `\end`.

```
2695 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2696 {
2697   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2698   { \@@_fatal:n { empty-environment } }
```

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

```
2699     \end { #2 }
2700 }
```

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).

```
2701 \cs_new:Npn \@@_create_col_nodes:
2702 {
2703   \crrc
2704   \int_compare:nNnT \l_@@_first_col_int = 0
2705   {
2706     \omit
2707     \hbox_overlap_left:n
2708     {
2709       \bool_if:NT \l_@@_code_before_bool
2710       { \pgfsys@markposition { \@@_env: - col - 0 } }
2711       \pgfpicture
2712       \pgfrememberpicturepositiononpagetrue
2713       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
2714       \str_if_empty:NF \l_@@_name_str
2715       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2716       \endpgfpicture
2717       \skip_horizontal:N 2\col@sep
2718       \skip_horizontal:N \g_@@_width_first_col_dim
2719     }
2720     &
2721   }
2722   \omit
```

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

```
2723   \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`).

```
2724   \int_compare:nNnTF \l_@@_first_col_int = 0
2725   {
2726     \bool_if:NT \l_@@_code_before_bool
2727     {
2728       \hbox
2729       {
2730         \skip_horizontal:N -0.5\arrayrulewidth
2731         \pgfsys@markposition { \@@_env: - col - 1 }
2732         \skip_horizontal:N 0.5\arrayrulewidth
2733       }
2734     }
2735     \pgfpicture
2736     \pgfrememberpicturepositiononpagetrue
2737     \pgfcoordinate { \@@_env: - col - 1 }
2738     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2739     \str_if_empty:NF \l_@@_name_str
2740     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2741     \endpgfpicture
2742   }
2743   {
2744     \bool_if:NT \l_@@_code_before_bool
2745     {
2746       \hbox
2747       {
2748         \skip_horizontal:N 0.5\arrayrulewidth
2749         \pgfsys@markposition { \@@_env: - col - 1 }
2750         \skip_horizontal:N -0.5\arrayrulewidth
```

```

2751     }
2752   }
2753   \pgfpicture
2754   \pgfrememberpicturepositiononpagetrue
2755   \pgfcoordinate { \@@_env: - col - 1 }
2756   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
2757   \str_if_empty:NF \l_@@_name_str
2758   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2759   \endpgfpicture
2760 }

```

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.

```

2761   \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2762   \bool_if:NF \l_@@_auto_columns_width_bool
2763   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2764   {
2765     \bool_lazy_and:nnTF
2766     \l_@@_auto_columns_width_bool
2767     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2768     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2769     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2770     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2771   }
2772   \skip_horizontal:N \g_tmpa_skip
2773   \hbox
2774   {
2775     \bool_if:NT \l_@@_code_before_bool
2776     {
2777       \hbox
2778       {
2779         \skip_horizontal:N -0.5\arrayrulewidth
2780         \pgfsys@markposition { \@@_env: - col - 2 }
2781         \skip_horizontal:N 0.5\arrayrulewidth
2782       }
2783     }
2784     \pgfpicture
2785     \pgfrememberpicturepositiononpagetrue
2786     \pgfcoordinate { \@@_env: - col - 2 }
2787     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2788     \str_if_empty:NF \l_@@_name_str
2789     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2790     \endpgfpicture
2791   }

```

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.

```

2792   \int_gset:Nn \g_tmpa_int 1
2793   \bool_if:NTF \g_@@_last_col_found_bool
2794   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2795   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2796   {
2797     &
2798     \omit
2799     \int_gincr:N \g_tmpa_int

```

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

```

2800     \skip_horizontal:N \g_tmpa_skip
2801     \bool_if:NT \l_@@_code_before_bool
2802     {

```



```

2803         \hbox
2804         {
2805             \skip_horizontal:N -0.5\arrayrulewidth
2806             \pgfsys@markposition
2807             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2808             \skip_horizontal:N 0.5\arrayrulewidth
2809         }
2810     }

```

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

```

2811     \pgfpicture
2812     \pgfrememberpicturepositiononpagetrue
2813     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2814     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2815     \str_if_empty:NF \l_@@_name_str
2816     {
2817         \pgfnodealias
2818         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
2819         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2820     }
2821     \endpgfpicture
2822 }

```

```

2823 &
2824 \omit

```

The two following lines have been added on 2021-12-15 to solve a bug mentionned by Joao Luis Soares by mail.

```

2825     \int_compare:nNnT \g_@@_col_total_int = 1
2826     { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
2827     \skip_horizontal:N \g_tmpa_skip
2828     \int_gincr:N \g_tmpa_int
2829     \bool_lazy_all:nT
2830     {
2831         \g_@@_NiceArray_bool
2832         { \bool_not_p:n \l_@@_NiceTabular_bool }
2833         { \clist_if_empty_p:N \l_@@_vlines_clist }
2834         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2835         { ! \l_@@_bar_at_end_of_pream_bool }
2836     }
2837     { \skip_horizontal:N -\col@sep }
2838     \bool_if:NT \l_@@_code_before_bool
2839     {
2840         \hbox
2841         {
2842             \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.

```

2843         \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
2844         { \skip_horizontal:N -\arraycolsep }
2845         \pgfsys@markposition
2846         { \@@_env: - col - \int_eval:n {
2847             \g_tmpa_int + 1 } }
2848         \skip_horizontal:N 0.5\arrayrulewidth
2849         \bool_lazy_and:nnT \l_@@_Matrix_bool \g_@@_NiceArray_bool
2850         { \skip_horizontal:N \arraycolsep }
2851     }
2852 }
2853 \pgfpicture
2854 \pgfrememberpicturepositiononpagetrue
2855 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2856 {

```

```

2857         \bool_lazy_and:nnTF \l_@@_Matrix_bool \g_@@_NiceArray_bool
2858         {
2859             \pgfpoint
2860             { - 0.5 \arrayrulewidth - \arraycolsep }
2861             \c_zero_dim
2862         }
2863         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2864     }
2865     \str_if_empty:NF \l_@@_name_str
2866     {
2867         \pgfnodealias
2868         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
2869         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2870     }
2871     \endpgfpicture

2872     \bool_if:NT \g_@@_last_col_found_bool
2873     {
2874         \hbox_overlap_right:n
2875         {
2876             \skip_horizontal:N \g_@@_width_last_col_dim
2877             \bool_if:NT \l_@@_code_before_bool
2878             {
2879                 \pgfsys@markposition
2880                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2881             }
2882             \pgfpicture
2883             \pgfrememberpicturepositiononpagetrue
2884             \pgfcoordinate
2885             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2886             \pgfpointorigin
2887             \str_if_empty:NF \l_@@_name_str
2888             {
2889                 \pgfnodealias
2890                 {
2891                     \l_@@_name_str - col
2892                     - \int_eval:n { \g_@@_col_total_int + 1 }
2893                 }
2894                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2895             }
2896             \endpgfpicture
2897         }
2898     }
2899     \cr
2900 }

```

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

```

2901 \tl_const:Nn \c_@@_preamble_first_col_tl
2902 {
2903     >
2904     {

```

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).

```

2905     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
2906     \bool_gset_true:N \g_@@_after_col_zero_bool
2907     \@@_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.

```

2908     \hbox_set:Nw \l_@@_cell_box
2909     \@@_math_toggle_token:
2910     \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”.

```

2911 \bool_lazy_and:nnT
2912 { \int_compare_p:nNn \c@iRow > 0 }
2913 {
2914   \bool_lazy_or_p:nn
2915   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2916   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2917 }
2918 {
2919   \l_@@_code_for_first_col_tl
2920   \xglobal \colorlet { nicematrix-first-col } { . }
2921 }
2922 }

```

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`.

```

2923 1
2924 <
2925 {
2926   \@@_math_toggle_token:
2927   \hbox_set_end:
2928   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2929   \@@_adjust_size_box:
2930   \@@_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.

```

2931 \dim_gset:Nn \g_@@_width_first_col_dim
2932 { \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.

```

2933 \hbox_overlap_left:n
2934 {
2935   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2936   \@@_node_for_cell:
2937   { \box_use_drop:N \l_@@_cell_box }
2938   \skip_horizontal:N \l_@@_left_delim_dim
2939   \skip_horizontal:N \l_@@_left_margin_dim
2940   \skip_horizontal:N \l_@@_extra_left_margin_dim
2941 }
2942 \bool_gset_false:N \g_@@_empty_cell_bool
2943 \skip_horizontal:N -2\col@sep
2944 }
2945 }

```

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

```

2946 \tl_const:Nn \c_@@_preamble_last_col_tl
2947 {
2948   >
2949   {

```

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).

```

2950 \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.

```

2951 \bool_gset_true:N \g_@@_last_col_found_bool
2952 \int_gincr:N \c@jCol
2953 \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.

```

2954 \hbox_set:Nw \l_@@_cell_box
2955 \@@_math_toggle_token:
2956 \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”.

```

2957     \int_compare:nNnT \c@iRow > 0
2958     {
2959         \bool_lazy_or:nnT
2960         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2961         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2962         {
2963             \l_@@_code_for_last_col_tl
2964             \xglobal \colorlet { nicematrix-last-col } { . }
2965         }
2966     }
2967 }
2968 1
2969 <
2970 {
2971     \@@_math_toggle_token:
2972     \hbox_set_end:
2973     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2974     \@@_adjust_size_box:
2975     \@@_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.

```

2976     \dim_gset:Nn \g_@@_width_last_col_dim
2977     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2978     \skip_horizontal:N -2\col@sep

```

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

```

2979     \hbox_overlap_right:n
2980     {
2981         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2982         {
2983             \skip_horizontal:N \l_@@_right_delim_dim
2984             \skip_horizontal:N \l_@@_right_margin_dim
2985             \skip_horizontal:N \l_@@_extra_right_margin_dim
2986             \@@_node_for_cell:
2987         }
2988     }
2989     \bool_gset_false:N \g_@@_empty_cell_bool
2990 }
2991 }

```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}` but, in fact, there is a flag `\g_@@_NiceArray_bool`. In `{NiceArrayWithDelims}`, some special code will be executed if this flag is raised.

```

2992 \NewDocumentEnvironment { NiceArray } { }
2993 {
2994     \bool_gset_true:N \g_@@_NiceArray_bool
2995     \str_if_empty:NT \g_@@_name_env_str
2996     { \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 `\g_@@_NiceArray_bool` is raised).

```

2997     \NiceArrayWithDelims . .
2998 }
2999 { \endNiceArrayWithDelims }

```

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

```

3000 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3001 {

```

```

3002 \NewDocumentEnvironment { #1 NiceArray } { }
3003 {
3004   \bool_gset_false:N \g_@@_NiceArray_bool
3005   \str_if_empty:NT \g_@@_name_env_str
3006   { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3007   \@@_test_if_math_mode:
3008   \NiceArrayWithDelims #2 #3
3009 }
3010 { \endNiceArrayWithDelims }
3011 }
3012 \@@_def_env:nnn p ( )
3013 \@@_def_env:nnn b [ ]
3014 \@@_def_env:nnn B \{ \}
3015 \@@_def_env:nnn v | |
3016 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

3017 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3018 {
3019   \bool_set_true:N \l_@@_Matrix_bool
3020   \use:c { #1 NiceArray }
3021   {
3022     *
3023     {
3024       \int_compare:nNnTF \l_@@_last_col_int < 0
3025       \c@MaxMatrixCols
3026       { \int_eval:n { \l_@@_last_col_int - 1 } }
3027     }
3028     { #2 }
3029   }
3030 }
3031 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3032 \clist_map_inline:nn { p , b , B , v , V }
3033 {
3034   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3035   {
3036     \bool_gset_false:N \g_@@_NiceArray_bool
3037     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3038     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3039     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3040   }
3041   { \use:c { end #1 NiceArray } }
3042 }

```

We define also an environment {NiceMatrix}

```

3043 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3044 {
3045   \bool_gset_false:N \g_@@_NiceArray_bool
3046   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3047   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3048   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3049 }
3050 { \endNiceArray }

```

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

```

3051 \cs_new_protected:Npn \@@_NotEmpty:
3052 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

{NiceTabular}, {NiceTabularX} and {NiceTabular*}

```

3053 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3054 {

```

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`.

```

3055   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3056     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3057   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3058   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3059   \bool_set_true:N \l_@@_NiceTabular_bool
3060   \NiceArray { #2 }
3061 }
3062 { \endNiceArray }

```

```

3063 \cs_set_protected:Npn \@@_newcolumnntype #1
3064 {
3065   \cs_if_free:cT { NC @ find @ #1 }
3066   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3067   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
3068   \peek_meaning:NTF [
3069     { \newcol@ #1 }
3070     { \newcol@ #1 [ 0 ] }
3071   }

```

```

3072 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3073 {

```

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}`).

```

3074   \bool_if:NT \c_@@_tabularx_loaded_bool { \newcolumnntype { X } { \@@_X } }
3075   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3076   \dim_zero_new:N \l_@@_width_dim
3077   \dim_set:Nn \l_@@_width_dim { #1 }
3078   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3079   \bool_set_true:N \l_@@_NiceTabular_bool
3080   \NiceArray { #3 }
3081 }
3082 { \endNiceArray }

```

```

3083 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
3084 {
3085   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3086   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3087   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3088   \bool_set_true:N \l_@@_NiceTabular_bool
3089   \NiceArray { #3 }
3090 }
3091 { \endNiceArray }

```

After the construction of the array

```

3092 \cs_new_protected:Npn \@@_after_array:
3093 {
3094   \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.

```

3095 \bool_if:NT \g_@@_last_col_found_bool
3096 { \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.

```

3097 \bool_if:NT \l_@@_last_col_without_value_bool
3098 { \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.

```

3099 \bool_if:NT \l_@@_last_row_without_value_bool
3100 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3101 \tl_gput_right:Nx \g_@@_aux_tl
3102 {
3103   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3104   {
3105     \int_use:N \l_@@_first_row_int ,
3106     \int_use:N \c@iRow ,
3107     \int_use:N \g_@@_row_total_int ,
3108     \int_use:N \l_@@_first_col_int ,
3109     \int_use:N \c@jCol ,
3110     \int_use:N \g_@@_col_total_int
3111   }
3112 }

```

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).

```

3113 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3114 {
3115   \tl_gput_right:Nx \g_@@_aux_tl
3116   {
3117     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3118     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3119   }
3120 }
3121 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3122 {
3123   \tl_gput_right:Nx \g_@@_aux_tl
3124   {
3125     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3126     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3127     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3128     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3129   }
3130 }

```

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

```

3131 \@@_create_diag_nodes:

```

We create the aliases using last for the nodes of the cells in the last row and the last column.

```

3132 \pgfpicture
3133 \int_step_inline:nn \c@iRow
3134 {
3135   \pgfnodealias
3136   { \@@_env: - ##1 - last }
3137   { \@@_env: - ##1 - \int_use:N \c@jCol }
3138 }
3139 \int_step_inline:nn \c@jCol
3140 {
3141   \pgfnodealias
3142   { \@@_env: - last - ##1 }
3143   { \@@_env: - \int_use:N \c@iRow - ##1 }
3144 }
3145 \str_if_empty:NF \l_@@_name_str

```

```

3146 {
3147   \int_step_inline:nn \c@iRow
3148   {
3149     \pgfnodealias
3150     { \l_@@_name_str - ##1 - last }
3151     { \@@_env: - ##1 - \int_use:N \c@jCol }
3152   }
3153   \int_step_inline:nn \c@jCol
3154   {
3155     \pgfnodealias
3156     { \l_@@_name_str - last - ##1 }
3157     { \@@_env: - \int_use:N \c@iRow - ##1 }
3158   }
3159 }
3160 \endpgfpicture

```

By default, the diagonal lines will be parallelized⁷⁰. 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.

```

3161 \bool_if:NT \l_@@_parallelize_diags_bool
3162 {
3163   \int_gzero_new:N \g_@@_ddots_int
3164   \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```

3165   \dim_gzero_new:N \g_@@_delta_x_one_dim
3166   \dim_gzero_new:N \g_@@_delta_y_one_dim
3167   \dim_gzero_new:N \g_@@_delta_x_two_dim
3168   \dim_gzero_new:N \g_@@_delta_y_two_dim
3169 }
3170 \int_zero_new:N \l_@@_initial_i_int
3171 \int_zero_new:N \l_@@_initial_j_int
3172 \int_zero_new:N \l_@@_final_i_int
3173 \int_zero_new:N \l_@@_final_j_int
3174 \bool_set_false:N \l_@@_initial_open_bool
3175 \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_xdots_radius_dim` and `\l_@@_xdots_inter_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.

```

3176 \bool_if:NT \l_@@_small_bool
3177 {
3178   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3179   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_end_dim` correspond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.

```

3180   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3181   { 0.6 \l_@@_xdots_shorten_start_dim }
3182   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3183   { 0.6 \l_@@_xdots_shorten_end_dim }
3184 }

```

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

```

3185 \@@_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.

⁷⁰It’s possible to use the option `parallelize-diags` to disable this parallelization.


```
3186 \@@_compute_corners:
```

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

```
3187 \@@_adjust_pos_of_blocks_seq:
3188 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3189 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3190 \bool_if:NT \c_@@_tikz_loaded_bool
3191 {
3192   \tikzset
3193   {
3194     every-picture / .style =
3195     {
3196       overlay ,
3197       remember-picture ,
3198       name-prefix = \@@_env: -
3199     }
3200   }
3201 }
3202 \cs_set_eq:NN \ialign \@@_old_ialign:
3203 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3204 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3205 \cs_set_eq:NN \OverBrace \@@_OverBrace
3206 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3207 \cs_set_eq:NN \line \@@_line
3208 \g_@@_internal_code_after_tl
3209 \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.

```
3210 \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).

```
3211 \seq_gclear:N \g_@@_submatrix_names_seq
```

And here’s the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```
3212 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3213 \scan_stop:
3214 \tl_gclear:N \g_nicematrix_code_after_tl
3215 \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.

```
3216 \tl_if_empty:NF \g_nicematrix_code_before_tl
3217 {
```

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`.

```
3218 \cs_set_protected:Npn \rectanglecolor { }
3219 \cs_set_protected:Npn \columncolor { }
3220 \tl_gput_right:Nx \g_@@_aux_tl
3221 {
3222   \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3223   { \exp_not:V \g_nicematrix_code_before_tl }
3224 }
```

```

3225     \bool_set_true:N \l_@@_code_before_bool
3226   }

```

```

3227   \str_gclear:N \g_@@_name_env_str
3228   \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁷¹. 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`.

```

3229   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3230 }

```

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`.

```

3231 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3232 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }

```

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

```

3233 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3234 {
3235   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3236   { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3237 }

```

The following command must *not* be protected.

```

3238 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3239 {
3240   { #1 }
3241   { #2 }
3242   {
3243     \int_compare:nNnTF { #3 } > { 99 }
3244     { \int_use:N \c@iRow }
3245     { #3 }
3246   }
3247   {
3248     \int_compare:nNnTF { #4 } > { 99 }
3249     { \int_use:N \c@jCol }
3250     { #4 }
3251   }
3252   { #5 }
3253 }

```

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).

```

3254 \hook_gput_code:nnn { begindocument } { . }
3255 {
3256   \cs_new_protected:Npx \@@_draw_dotted_lines:
3257   {
3258     \c_@@_pgfortikzpicture_tl
3259     \@@_draw_dotted_lines_i:

```

⁷¹e.g. `\color[rgb]{0.5,0.5,0}`

```

3260         \c_@@_endpgfortikzpicture_tl
3261     }
3262 }

```

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

```

3263 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3264 {
3265     \pgfrememberpicturepositiononpagetrue
3266     \pgf@relevantforpicturesizefalse
3267     \g_@@_HVdotsfor_lines_tl
3268     \g_@@_Vdots_lines_tl
3269     \g_@@_Ddots_lines_tl
3270     \g_@@_Iddots_lines_tl
3271     \g_@@_Cdots_lines_tl
3272     \g_@@_Ldots_lines_tl
3273 }

3274 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3275 {
3276     \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3277     \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3278 }

```

We define a new PGF shape for the diag nodes because we want to provide a anchor called `.5` for those nodes.

```

3279 \pgfdeclareshape { @@_diag_node }
3280 {
3281     \savedanchor { \five }
3282     {
3283         \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3284         \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3285     }
3286     \anchor { 5 } { \five }
3287     \anchor { center } { \pgfpointorigin }
3288 }

```

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).

```

3289 \cs_new_protected:Npn \@@_create_diag_nodes:
3290 {
3291     \pgfpicture
3292     \pgfrememberpicturepositiononpagetrue
3293     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3294     {
3295         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3296         \dim_set_eq:NN \l_tmpa_dim \pgf@x
3297         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3298         \dim_set_eq:NN \l_tmpb_dim \pgf@y
3299         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3300         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3301         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3302         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3303         \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 `@@_diag_node`) that we will construct.

```

3304         \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3305         \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3306         \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3307         \str_if_empty:NF \l_@@_name_str
3308         { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3309     }

```

Now, the last node. Of course, that is only a `coordinate` because there is not `.5` anchor for that node.

```

3310 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3311 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3312 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3313 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3314 \pgfcoordinate
3315   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3316 \pgfnodealias
3317   { \@@_env: - last }
3318   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3319 \str_if_empty:NF \l_@@_name_str
3320 {
3321   \pgfnodealias
3322     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3323     { \@@_env: - \int_use:N \l_tmpa_int }
3324   \pgfnodealias
3325     { \l_@@_name_str - last }
3326     { \@@_env: - last }
3327 }
3328 \endpgfpicture
3329 }

```

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.

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

The command `\@@ find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
3330 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3331 {
```

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

```
3332 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3333 \int_set:Nn \l_@@_initial_i_int { #1 }
3334 \int_set:Nn \l_@@_initial_j_int { #2 }
3335 \int_set:Nn \l_@@_final_i_int { #1 }
3336 \int_set:Nn \l_@@_final_j_int { #2 }
```

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

```

3337     \bool_set_false:N \l_@@_stop_loop_bool
3338     \bool_do_until:Nn \l_@@_stop_loop_bool
3339     {
3340         \int_add:Nn \l_@@_final_i_int { #3 }
3341         \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3342     \bool_set_false:N \l_@@_final_open_bool
3343     \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3344     {
3345         \int_compare:nNnTF { #3 } = 1
3346         { \bool_set_true:N \l_@@_final_open_bool }
3347         {
3348             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3349             { \bool_set_true:N \l_@@_final_open_bool }
3350         }
3351     }
3352     {
3353         \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3354         {
3355             \int_compare:nNnT { #4 } = { -1 }
3356             { \bool_set_true:N \l_@@_final_open_bool }
3357         }
3358         {
3359             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3360             {
3361                 \int_compare:nNnT { #4 } = 1
3362                 { \bool_set_true:N \l_@@_final_open_bool }
3363             }
3364         }
3365     }
3366     \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.

```

3367     {

```

We do a step backwards.

```

3368         \int_sub:Nn \l_@@_final_i_int { #3 }
3369         \int_sub:Nn \l_@@_final_j_int { #4 }
3370         \bool_set_true:N \l_@@_stop_loop_bool
3371     }

```

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`.

```

3372     {
3373         \cs_if_exist:cTF
3374         {
3375             @@ _ dotted _
3376             \int_use:N \l_@@_final_i_int -
3377             \int_use:N \l_@@_final_j_int
3378         }
3379         {
3380             \int_sub:Nn \l_@@_final_i_int { #3 }
3381             \int_sub:Nn \l_@@_final_j_int { #4 }
3382             \bool_set_true:N \l_@@_final_open_bool
3383             \bool_set_true:N \l_@@_stop_loop_bool
3384         }
3385         {
3386             \cs_if_exist:cTF
3387             {
3388                 pgf @ sh @ ns @ \@@_env:
3389                 - \int_use:N \l_@@_final_i_int

```

```

3390         - \int_use:N \l_@@_final_j_int
3391     }
3392     { \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.

```

3393     {
3394         \cs_set:cpn
3395         {
3396             @@ _ dotted _
3397             \int_use:N \l_@@_final_i_int -
3398             \int_use:N \l_@@_final_j_int
3399         }
3400         { }
3401     }
3402 }
3403 }
3404 }

```

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

```

3405     \bool_set_false:N \l_@@_stop_loop_bool
3406     \bool_do_until:Nn \l_@@_stop_loop_bool
3407     {
3408         \int_sub:Nn \l_@@_initial_i_int { #3 }
3409         \int_sub:Nn \l_@@_initial_j_int { #4 }
3410         \bool_set_false:N \l_@@_initial_open_bool
3411         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3412         {
3413             \int_compare:nNnTF { #3 } = 1
3414             { \bool_set_true:N \l_@@_initial_open_bool }
3415             {
3416                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3417                 { \bool_set_true:N \l_@@_initial_open_bool }
3418             }
3419         }
3420         {
3421             \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3422             {
3423                 \int_compare:nNnT { #4 } = 1
3424                 { \bool_set_true:N \l_@@_initial_open_bool }
3425             }
3426             {
3427                 \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3428                 {
3429                     \int_compare:nNnT { #4 } = { -1 }
3430                     { \bool_set_true:N \l_@@_initial_open_bool }
3431                 }
3432             }
3433         }
3434         \bool_if:NnTF \l_@@_initial_open_bool
3435         {
3436             \int_add:Nn \l_@@_initial_i_int { #3 }
3437             \int_add:Nn \l_@@_initial_j_int { #4 }
3438             \bool_set_true:N \l_@@_stop_loop_bool
3439         }
3440         {
3441             \cs_if_exist:cTF
3442             {

```

```

3443         @@ _ dotted _
3444         \int_use:N \l_@@_initial_i_int -
3445         \int_use:N \l_@@_initial_j_int
3446     }
3447     {
3448         \int_add:Nn \l_@@_initial_i_int { #3 }
3449         \int_add:Nn \l_@@_initial_j_int { #4 }
3450         \bool_set_true:N \l_@@_initial_open_bool
3451         \bool_set_true:N \l_@@_stop_loop_bool
3452     }
3453     {
3454         \cs_if_exist:cTF
3455         {
3456             pgf @ sh @ ns @ \@@_env:
3457             - \int_use:N \l_@@_initial_i_int
3458             - \int_use:N \l_@@_initial_j_int
3459         }
3460         { \bool_set_true:N \l_@@_stop_loop_bool }
3461         {
3462             \cs_set:cpn
3463             {
3464                 @@ _ dotted _
3465                 \int_use:N \l_@@_initial_i_int -
3466                 \int_use:N \l_@@_initial_j_int
3467             }
3468             { }
3469         }
3470     }
3471 }
3472 }

```

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.

```

3473     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3474     {
3475         { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

3476         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3477         { \int_use:N \l_@@_final_i_int }
3478         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3479         { } % for the name of the block
3480     }
3481 }

```

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).

```

3482 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3483 {
3484     \int_set:Nn \l_@@_row_min_int 1
3485     \int_set:Nn \l_@@_col_min_int 1
3486     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3487     \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`.

```

3488     \seq_map_inline:Nn \g_@@_submatrix_seq
3489     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
3490 }

```

#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 we are analyzing.

```

3491 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3492 {
3493   \bool_if:nT
3494   {
3495     \int_compare_p:n { #3 <= #1 }
3496     && \int_compare_p:n { #1 <= #5 }
3497     && \int_compare_p:n { #4 <= #2 }
3498     && \int_compare_p:n { #2 <= #6 }
3499   }
3500   {
3501     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3502     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3503     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3504     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3505   }
3506 }

3507 \cs_new_protected:Npn \@@_set_initial_coords:
3508 {
3509   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3510   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3511 }
3512 \cs_new_protected:Npn \@@_set_final_coords:
3513 {
3514   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3515   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3516 }
3517 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3518 {
3519   \pgfpointanchor
3520   {
3521     \@@_env:
3522     - \int_use:N \l_@@_initial_i_int
3523     - \int_use:N \l_@@_initial_j_int
3524   }
3525   { #1 }
3526   \@@_set_initial_coords:
3527 }
3528 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3529 {
3530   \pgfpointanchor
3531   {
3532     \@@_env:
3533     - \int_use:N \l_@@_final_i_int
3534     - \int_use:N \l_@@_final_j_int
3535   }
3536   { #1 }
3537   \@@_set_final_coords:
3538 }

3539 \cs_new_protected:Npn \@@_open_x_initial_dim:
3540 {
3541   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3542   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3543   {
3544     \cs_if_exist:cT
3545     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3546     {
3547       \pgfpointanchor
3548       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3549       { west }

```



```

3550         \dim_set:Nn \l_@@_x_initial_dim
3551         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3552     }
3553 }

```

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

```

3554     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3555     {
3556         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3557         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3558         \dim_add:Nn \l_@@_x_initial_dim \col@sep
3559     }
3560 }
3561 \cs_new_protected:Npn \@@_open_x_final_dim:
3562 {
3563     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3564     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3565     {
3566         \cs_if_exist:cT
3567         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3568         {
3569             \pgfpointanchor
3570             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3571             { east }
3572             \dim_set:Nn \l_@@_x_final_dim
3573             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3574         }
3575     }

```

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

```

3576     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3577     {
3578         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
3579         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3580         \dim_sub:Nn \l_@@_x_final_dim \col@sep
3581     }
3582 }

```

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.

```

3583 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3584 {
3585     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3586     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3587     {
3588         \@@_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.

```

3589     \group_begin:
3590     \int_compare:nNnTF { #1 } = 0
3591     { \color { nicematrix-first-row } }
3592     {

```

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.

```

3593         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3594         { \color { nicematrix-last-row } }
3595     }
3596     \keys_set:nn { NiceMatrix / xdots } { #3 }
3597     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3598     \@@_actually_draw_Ldots:
3599     \group_end:

```

```

3600     }
3601 }

```

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`.

```

3602 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3603 {
3604   \bool_if:NTF \l_@@_initial_open_bool
3605   {
3606     \@@_open_x_initial_dim:
3607     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3608     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3609   }
3610   { \@@_set_initial_coords_from_anchor:n { base-east } }
3611   \bool_if:NTF \l_@@_final_open_bool
3612   {
3613     \@@_open_x_final_dim:
3614     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3615     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3616   }
3617   { \@@_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 (?).

```

3618   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
3619   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
3620   \@@_draw_line:
3621 }

```

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.

```

3622 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3623 {
3624   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3625   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3626   {
3627     \@@_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.

```

3628   \group_begin:
3629   \int_compare:nNnTF { #1 } = 0
3630   { \color { nicematrix-first-row } }
3631   {

```

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.

```

3632     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3633     { \color { nicematrix-last-row } }
3634   }
3635   \keys_set:nn { NiceMatrix / xdots } { #3 }

```

```

3636         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3637         \@@_actually_draw_Cdots:
3638     \group_end:
3639 }
3640 }

```

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.`

```

3641 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3642 {
3643     \bool_if:NTF \l_@@_initial_open_bool
3644     { \@@_open_x_initial_dim: }
3645     { \@@_set_initial_coords_from_anchor:n { mid-east } }
3646     \bool_if:NTF \l_@@_final_open_bool
3647     { \@@_open_x_final_dim: }
3648     { \@@_set_final_coords_from_anchor:n { mid-west } }
3649     \bool_lazy_and:nnTF
3650     \l_@@_initial_open_bool
3651     \l_@@_final_open_bool
3652     {
3653         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3654         \dim_set_eq:NN \l_tmpa_dim \pgf@y
3655         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
3656         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3657         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3658     }
3659     {
3660         \bool_if:NT \l_@@_initial_open_bool
3661         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3662         \bool_if:NT \l_@@_final_open_bool
3663         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3664     }
3665     \@@_draw_line:
3666 }
3667 \cs_new_protected:Npn \@@_open_y_initial_dim:
3668 {
3669     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3670     \dim_set:Nn \l_@@_y_initial_dim
3671     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3672     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3673     {
3674         \cs_if_exist:cT
3675         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3676         {
3677             \pgfpointanchor
3678             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3679             { north }
3680             \dim_set:Nn \l_@@_y_initial_dim
3681             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3682         }
3683     }
3684 }

```

```

3685 \cs_new_protected:Npn \@@_open_y_final_dim:
3686 {
3687   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3688   \dim_set:Nn \l_@@_y_final_dim
3689   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3690   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3691   {
3692     \cs_if_exist:cT
3693     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3694     {
3695       \pgfpointanchor
3696       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3697       { south }
3698       \dim_set:Nn \l_@@_y_final_dim
3699       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3700     }
3701   }
3702 }

```

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.

```

3703 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3704 {
3705   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3706   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3707   {
3708     \@@_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.

```

3709   \group_begin:
3710   \int_compare:nNnTF { #2 } = 0
3711   { \color { nicematrix-first-col } }
3712   {
3713     \int_compare:nNnT { #2 } = \l_@@_last_col_int
3714     { \color { nicematrix-last-col } }
3715   }
3716   \keys_set:nn { NiceMatrix / xdots } { #3 }
3717   \tl_if_empty:VF \l_@@_xdots_color_tl
3718   { \color { \l_@@_xdots_color_tl } }
3719   \@@_actually_draw_Vdots:
3720 \group_end:
3721 }
3722 }

```

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`.

```

3723 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3724 {

```

The boolean `\l_tmpa_bool` indicates whether the column is of type 1 or may be considered as if.

```

3725   \bool_set_false:N \l_tmpa_bool

```

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

```

3726 \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3727 {
3728   \@@_set_initial_coords_from_anchor:n { south-west }
3729   \@@_set_final_coords_from_anchor:n { north-west }
3730   \bool_set:Nn \l_tmpa_bool
3731     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3732 }

```

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

```

3733 \bool_if:NTF \l_@@_initial_open_bool
3734   \@@_open_y_initial_dim:
3735   { \@@_set_initial_coords_from_anchor:n { south } }
3736 \bool_if:NTF \l_@@_final_open_bool
3737   \@@_open_y_final_dim:
3738   { \@@_set_final_coords_from_anchor:n { north } }
3739 \bool_if:NTF \l_@@_initial_open_bool
3740 {
3741   \bool_if:NTF \l_@@_final_open_bool
3742   {
3743     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3744     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3745     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
3746     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3747     \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.

```

3748 \int_compare:nNnT \l_@@_last_col_int > { -2 }
3749 {
3750   \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3751   {
3752     \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3753     \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3754     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3755     \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3756   }
3757 }
3758 }
3759 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3760 }
3761 {
3762   \bool_if:NTF \l_@@_final_open_bool
3763   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3764 }

```

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.

```

3765 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3766 {
3767   \dim_set:Nn \l_@@_x_initial_dim
3768   {
3769     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3770     \l_@@_x_initial_dim \l_@@_x_final_dim
3771   }
3772   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3773 }
3774 }
3775 }
3776 \@@_draw_line:
3777 }

```

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.

```

3778 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3779 {
3780   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3781   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3782   {
3783     \@@_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.

```

3784     \group_begin:
3785     \keys_set:nn { NiceMatrix / xdots } { #3 }
3786     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3787     \@@_actually_draw_Ddots:
3788   \group_end:
3789 }
3790 }

```

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`.

```

3791 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3792 {
3793   \bool_if:NTF \l_@@_initial_open_bool
3794   {
3795     \@@_open_y_initial_dim:
3796     \@@_open_x_initial_dim:
3797   }
3798   { \@@_set_initial_coords_from_anchor:n { south-east } }
3799   \bool_if:NTF \l_@@_final_open_bool
3800   {
3801     \@@_open_x_final_dim:
3802     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3803   }
3804   { \@@_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.

```

3805   \bool_if:NT \l_@@_parallelize_diags_bool
3806   {
3807     \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).

```

3808     \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.

```

3809     {
3810       \dim_gset:Nn \g_@@_delta_x_one_dim

```

```

3811         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3812         \dim_gset:Nn \g_@@_delta_y_one_dim
3813         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3814     }

```

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`.

```

3815     {
3816         \dim_set:Nn \l_@@_y_final_dim
3817         {
3818             \l_@@_y_initial_dim +
3819             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3820             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3821         }
3822     }
3823 }
3824 \@@_draw_line:
3825 }

```

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.

```

3826 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3827 {
3828     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3829     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3830     {
3831         \@@_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.

```

3832     \group_begin:
3833     \keys_set:nn { NiceMatrix / xdots } { #3 }
3834     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3835     \@@_actually_draw_Iddots:
3836     \group_end:
3837 }
3838 }

```

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`.

```

3839 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3840 {
3841     \bool_if:NTF \l_@@_initial_open_bool
3842     {
3843         \@@_open_y_initial_dim:
3844         \@@_open_x_initial_dim:
3845     }
3846     { \@@_set_initial_coords_from_anchor:n { south-west } }
3847     \bool_if:NTF \l_@@_final_open_bool
3848     {
3849         \@@_open_y_final_dim:
3850         \@@_open_x_final_dim:

```

```

3851     }
3852     { \@@_set_final_coords_from_anchor:n { north~east } }
3853     \bool_if:NT \l_@@_parallelize_diags_bool
3854     {
3855         \int_gincr:N \g_@@_iddots_int
3856         \int_compare:nNnTF \g_@@_iddots_int = 1
3857         {
3858             \dim_gset:Nn \g_@@_delta_x_two_dim
3859             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3860             \dim_gset:Nn \g_@@_delta_y_two_dim
3861             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3862         }
3863         {
3864             \dim_set:Nn \l_@@_y_final_dim
3865             {
3866                 \l_@@_y_initial_dim +
3867                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3868                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3869             }
3870         }
3871     }
3872     \@@_draw_line:
3873 }

```

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`

```

3874 \cs_new_protected:Npn \@@_draw_line:
3875 {
3876     \pgfrememberpicturepositiononpagetrue
3877     \pgf@relevantforpicturesizefalse
3878     \bool_lazy_or:nnTF
3879     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }

```

The boolean `\l_@@_dotted_bool` is raised for the rules specified by either `\hdottedline` or `:` (or the letter specified by `letter-for-dotted-lines`) in the preamble of the array.

```

3880     \l_@@_dotted_bool
3881     \@@_draw_standard_dotted_line:
3882     \@@_draw_unstandard_dotted_line:
3883 }

```

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.

```

3884 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
3885 {
3886     \begin { scope }
3887     \@@_draw_unstandard_dotted_line:o
3888     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3889 }

```


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.

```

3890 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
3891 {
3892   \@@_draw_unstandard_dotted_line:nVV
3893   { #1 }
3894   \l_@@_xdots_up_tl
3895   \l_@@_xdots_down_tl
3896 }
3897 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
3898 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
3899 {
3900   \draw
3901   [
3902     #1 ,
3903     shorten-> = \l_@@_xdots_shorten_end_dim ,
3904     shorten-< = \l_@@_xdots_shorten_start_dim ,
3905   ]
3906   ( \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).

```

3907   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3908   node [ sloped , below ] { $ \scriptstyle #3 $ }
3909   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3910 \end { scope }
3911 }
3912 \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).

```

3913 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3914 {
3915   \bool_lazy_and:nnF
3916   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3917   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3918   {
3919     \pgfscope
3920     \pgftransformshift
3921     {
3922       \pgfpointlineattime { 0.5 }
3923       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3924       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3925     }
3926     \pgftransformrotate
3927     {
3928       \fp_eval:n
3929       {
3930         atand
3931         (
3932           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3933           \l_@@_x_final_dim - \l_@@_x_initial_dim
3934         )
3935       }
3936     }
3937     \pgfnode
3938     { rectangle }
3939     { south }
3940     {
3941       \c_math_toggle_token
3942       \scriptstyle \l_@@_xdots_up_tl

```

```

3943         \c_math_toggle_token
3944     }
3945     { }
3946     { \pgfusepath { } }
3947 \pgfnode
3948     { rectangle }
3949     { north }
3950     {
3951         \c_math_toggle_token
3952         \scriptstyle \l_@@_xdots_down_tl
3953         \c_math_toggle_token
3954     }
3955     { }
3956     { \pgfusepath { } }
3957 \endpgfscope
3958 }
3959 \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.

```

3960     \dim_zero_new:N \l_@@_l_dim
3961     \dim_set:Nn \l_@@_l_dim
3962     {
3963         \fp_to_dim:n
3964         {
3965             sqrt
3966             (
3967                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3968                 +
3969                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3970             )
3971         }
3972     }

```

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.

```

3973     \bool_lazy_or:nnF
3974     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3975     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3976     \@@_draw_standard_dotted_line_i:
3977 \group_end:
3978 }
3979 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3980 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3981 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

3982     \bool_if:NTF \l_@@_initial_open_bool
3983     {
3984         \bool_if:NTF \l_@@_final_open_bool
3985         {
3986             \int_set:Nn \l_tmpa_int
3987             { \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim }
3988         }
3989         {
3990             \int_set:Nn \l_tmpa_int
3991             {
3992                 \dim_ratio:nn
3993                 { \l_@@_l_dim - \l_@@_xdots_shorten_start_dim }
3994                 \l_@@_xdots_inter_dim
3995             }

```

```

3996     }
3997   }
3998   {
3999     \bool_if:NTF \l_@@_final_open_bool
4000     {
4001       \int_set:Nn \l_tmpa_int
4002       {
4003         \dim_ratio:nn
4004         { \l_@@_l_dim - \l_@@_xdots_shorten_end_dim }
4005         \l_@@_xdots_inter_dim
4006       }
4007     }
4008     {
4009       \int_set:Nn \l_tmpa_int
4010       {
4011         \dim_ratio:nn
4012         {
4013           \l_@@_l_dim
4014           - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4015         }
4016         \l_@@_xdots_inter_dim
4017       }
4018     }
4019   }

```

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

```

4020   \dim_set:Nn \l_tmpa_dim
4021   {
4022     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4023     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4024   }
4025   \dim_set:Nn \l_tmpb_dim
4026   {
4027     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4028     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4029   }

```

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.

```

4030   \dim_gadd:Nn \l_@@_x_initial_dim
4031   {
4032     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4033     \dim_ratio:nn
4034     {
4035       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4036       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4037     }
4038     { 2 \l_@@_l_dim }
4039   }
4040   \dim_gadd:Nn \l_@@_y_initial_dim
4041   {
4042     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4043     \dim_ratio:nn
4044     {
4045       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4046       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4047     }
4048     { 2 \l_@@_l_dim }
4049   }
4050   \pgf@relevantforpicturesizefalse
4051   \int_step_inline:nnn 0 \l_tmpa_int
4052   {
4053     \pgfpathcircle

```

```

4054         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4055         { \l_@@_xdots_radius_dim }
4056         \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4057         \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4058     }
4059     \pgfusepathqfill
4060 }

```

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 `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

4061 \hook_gput_code:nnn { begindocument } { . }
4062 {
4063     \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
4064     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4065     \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
4066     {
4067         \int_compare:nNnTF \c@jCol = 0
4068         { \@@_error:nn { in~first~col } \Ldots }
4069         {
4070             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4071             { \@@_error:nn { in~last~col } \Ldots }
4072             {
4073                 \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4074                 { #1 , down = #2 , up = #3 }
4075             }
4076         }
4077         \bool_if:NF \l_@@_nullify_dots_bool
4078         { \phantom { \ensuremath { \@@_old_ldots } } }
4079         \bool_gset_true:N \g_@@_empty_cell_bool
4080     }

4081     \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4082     {
4083         \int_compare:nNnTF \c@jCol = 0
4084         { \@@_error:nn { in~first~col } \Cdots }
4085         {
4086             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4087             { \@@_error:nn { in~last~col } \Cdots }
4088             {
4089                 \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4090                 { #1 , down = #2 , up = #3 }
4091             }
4092         }
4093         \bool_if:NF \l_@@_nullify_dots_bool
4094         { \phantom { \ensuremath { \@@_old_cdots } } }
4095         \bool_gset_true:N \g_@@_empty_cell_bool
4096     }

```

```

4097 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4098 {
4099   \int_compare:nNnTF \c@iRow = 0
4100   { \@@_error:nn { in~first~row } \Vdots }
4101   {
4102     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4103     { \@@_error:nn { in~last~row } \Vdots }
4104     {
4105       \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4106       { #1 , down = #2 , up = #3 }
4107     }
4108   }
4109   \bool_if:NF \l_@@_nullify_dots_bool
4110   { \phantom { \ensuremath { \@@_old_vdots } } }
4111   \bool_gset_true:N \g_@@_empty_cell_bool
4112 }

4113 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4114 {
4115   \int_case:nnF \c@iRow
4116   {
4117     0 { \@@_error:nn { in~first~row } \Ddots }
4118     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4119   }
4120   {
4121     \int_case:nnF \c@jCol
4122     {
4123       0 { \@@_error:nn { in~first~col } \Ddots }
4124       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4125     }
4126     {
4127       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4128       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
4129       { #1 , down = #2 , up = #3 }
4130     }
4131   }
4132 }
4133 \bool_if:NF \l_@@_nullify_dots_bool
4134 { \phantom { \ensuremath { \@@_old_ddots } } }
4135 \bool_gset_true:N \g_@@_empty_cell_bool
4136 }

4137 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4138 {
4139   \int_case:nnF \c@iRow
4140   {
4141     0 { \@@_error:nn { in~first~row } \Iddots }
4142     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4143   }
4144   {
4145     \int_case:nnF \c@jCol
4146     {
4147       0 { \@@_error:nn { in~first~col } \Iddots }
4148       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4149     }
4150     {
4151       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4152       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Iddots }
4153       { #1 , down = #2 , up = #3 }
4154     }
4155   }
4156   \bool_if:NF \l_@@_nullify_dots_bool

```

```

4157         { \phantom { \ensuremath { \@@_old_iddots } } }
4158         \bool_gset_true:N \g_@@_empty_cell_bool
4159     }
4160 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

4161 \keys_define:nn { NiceMatrix / Ddots }
4162 {
4163     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4164     draw-first .default:n = true ,
4165     draw-first .value_forbidden:n = true
4166 }

```

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

```

4167 \cs_new_protected:Npn \@@_Hspace:
4168 {
4169     \bool_gset_true:N \g_@@_empty_cell_bool
4170     \hspace
4171 }

```

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.

```

4172 \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.

```

4173 \cs_new:Npn \@@_Hdotsfor:
4174 {
4175     \bool_lazy_and:nnTF
4176     { \int_compare_p:nNn \c@jCol = 0 }
4177     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4178     {
4179         \bool_if:NTF \g_@@_after_col_zero_bool
4180         {
4181             \multicolumn { 1 } { c } { }
4182             \@@_Hdotsfor_i
4183         }
4184         { \@@_fatal:n { Hdotsfor~in~col~0 } }
4185     }
4186     {
4187         \multicolumn { 1 } { c } { }
4188         \@@_Hdotsfor_i
4189     }
4190 }

```

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:).

```

4191 \hook_gput_code:nnn { begindocument } { . }
4192 {
4193     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4194     \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.

```

4195     \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4196     {
4197         \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl

```

```

4198     {
4199         \@@_Hdotsfor:nnnn
4200         { \int_use:N \c@iRow }
4201         { \int_use:N \c@jCol }
4202         { #2 }
4203         {
4204             #1 , #3 ,
4205             down = \exp_not:n { #4 } ,
4206             up = \exp_not:n { #5 }
4207         }
4208     }
4209     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4210 }
4211 }

```

Enf of \AddToHook.

```

4212 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4213 {
4214     \bool_set_false:N \l_@@_initial_open_bool
4215     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4216     \int_set:Nn \l_@@_initial_i_int { #1 }
4217     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4218     \int_compare:nNnTF { #2 } = 1
4219     {
4220         \int_set:Nn \l_@@_initial_j_int 1
4221         \bool_set_true:N \l_@@_initial_open_bool
4222     }
4223     {
4224         \cs_if_exist:cTF
4225         {
4226             pgf @ sh @ ns @ \@@_env:
4227             - \int_use:N \l_@@_initial_i_int
4228             - \int_eval:n { #2 - 1 }
4229         }
4230         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4231         {
4232             \int_set:Nn \l_@@_initial_j_int { #2 }
4233             \bool_set_true:N \l_@@_initial_open_bool
4234         }
4235     }
4236     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4237     {
4238         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4239         \bool_set_true:N \l_@@_final_open_bool
4240     }
4241     {
4242         \cs_if_exist:cTF
4243         {
4244             pgf @ sh @ ns @ \@@_env:
4245             - \int_use:N \l_@@_final_i_int
4246             - \int_eval:n { #2 + #3 }
4247         }
4248         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4249         {
4250             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4251             \bool_set_true:N \l_@@_final_open_bool
4252         }
4253     }
4254     \group_begin:

```

```

4255 \int_compare:nNnTF { #1 } = 0
4256 { \color { nicematrix-first-row } }
4257 {
4258   \int_compare:nNnT { #1 } = \g_@@_row_total_int
4259   { \color { nicematrix-last-row } }
4260 }
4261 \keys_set:nn { NiceMatrix / xdots } { #4 }
4262 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4263 \@@_actually_draw_Ldots:
4264 \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).

```

4265 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4266 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4267 }

4268 \hook_gput_code:nnn { begindocument } { . }
4269 {
4270   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4271   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4272   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4273   {
4274     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4275     {
4276       \@@_Vdotsfor:nnnn
4277       { \int_use:N \c@iRow }
4278       { \int_use:N \c@jCol }
4279       { #2 }
4280       {
4281         #1 , #3 ,
4282         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4283       }
4284     }
4285   }
4286 }

```

Enf of `\AddToHook`.

```

4287 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4288 {
4289   \bool_set_false:N \l_@@_initial_open_bool
4290   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

4291 \int_set:Nn \l_@@_initial_j_int { #2 }
4292 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

4293 \int_compare:nNnTF #1 = 1
4294 {
4295   \int_set:Nn \l_@@_initial_i_int 1
4296   \bool_set_true:N \l_@@_initial_open_bool
4297 }
4298 {
4299   \cs_if_exist:cTF
4300   {
4301     pgf @ sh @ ns @ \@@_env:
4302     - \int_eval:n { #1 - 1 }
4303     - \int_use:N \l_@@_initial_j_int
4304   }
4305   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4306 }

```



```

4307         \int_set:Nn \l_@@_initial_i_int { #1 }
4308         \bool_set_true:N \l_@@_initial_open_bool
4309     }
4310 }
4311 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4312 {
4313     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4314     \bool_set_true:N \l_@@_final_open_bool
4315 }
4316 {
4317     \cs_if_exist:cTF
4318     {
4319         pgf @ sh @ ns @ \@@_env:
4320         - \int_eval:n { #1 + #3 }
4321         - \int_use:N \l_@@_final_j_int
4322     }
4323     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4324     {
4325         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4326         \bool_set_true:N \l_@@_final_open_bool
4327     }
4328 }
4329 \group_begin:
4330 \int_compare:nNnTF { #2 } = 0
4331 { \color { nicematrix-first-col } }
4332 {
4333     \int_compare:nNnT { #2 } = \g_@@_col_total_int
4334     { \color { nicematrix-last-col } }
4335 }
4336 \keys_set:nn { NiceMatrix / xdots } { #4 }
4337 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4338 \@@_actually_draw_Vdots:
4339 \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).

```

4340     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4341     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4342 }

```

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

```

4343 \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 the following behaviour:

- If the argument is of the format i - j , our command applies the command `\int_eval:n` to i and j ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).⁷²

```

4344 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4345 {
4346   \tl_if_empty:nTF { #2 }
4347     { #1 }
4348     { \@@_double_int_eval_i:n #1-#2 \q_stop }
4349 }
4350 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
4351 { \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).

```

4352 \hook_gput_code:nnn { begindocument } { . }
4353 {
4354   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4355   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4356   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4357     {
4358       \group_begin:
4359       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4360       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4361       \use:e
4362       {
4363         \@@_line_i:nn
4364         { \@@_double_int_eval:n #2 - \q_stop }
4365         { \@@_double_int_eval:n #3 - \q_stop }
4366       }
4367       \group_end:
4368     }
4369 }
4370 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4371 {
4372   \bool_set_false:N \l_@@_initial_open_bool
4373   \bool_set_false:N \l_@@_final_open_bool
4374   \bool_if:nTF
4375     {
4376       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4377       ||
4378       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4379     }
4380     {
4381       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4382     }
4383     { \@@_draw_line_ii:nn { #1 } { #2 } }
4384 }
4385 \hook_gput_code:nnn { begindocument } { . }
4386 {
4387   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4388   {

```

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:.`

```

4389   \c_@@_pgfortikzpicture_tl
4390   \@@_draw_line_iii:nn { #1 } { #2 }
4391   \c_@@_endpgfortikzpicture_tl
4392 }
4393 }

```

⁷²Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

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

```

4394 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4395 {
4396   \pgfrememberpicturepositiononpagetrue
4397   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4398   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4399   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4400   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4401   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4402   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4403   \@@_draw_line:
4404 }

```

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`

```

4405 \keys_define:nn { NiceMatrix / RowStyle }
4406 {
4407   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4408   cell-space-top-limit .initial:n = \c_zero_dim ,
4409   cell-space-top-limit .value_required:n = true ,
4410   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4411   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4412   cell-space-bottom-limit .value_required:n = true ,
4413   cell-space-limits .meta:n =
4414   {
4415     cell-space-top-limit = #1 ,
4416     cell-space-bottom-limit = #1 ,
4417   } ,
4418   color .tl_set:N = \l_@@_color_tl ,
4419   color .value_required:n = true ,
4420   bold .bool_set:N = \l_tmpa_bool ,
4421   bold .default:n = true ,
4422   bold .initial:n = false ,
4423   nb-rows .code:n =
4424   \str_if_eq:nnTF { #1 } { * }
4425   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
4426   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
4427   nb-rows .value_required:n = true ,
4428   rowcolor .tl_set:N = \l_tmpa_tl ,
4429   rowcolor .value_required:n = true ,
4430   rowcolor .initial:n = ,
4431   unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
4432 }

4433 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4434 {
4435   \group_begin:
4436   \tl_clear:N \l_@@_color_tl
4437   \int_set:Nn \l_@@_key_nb_rows_int 1
4438   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

4439   \tl_if_empty:NF \l_tmpa_tl
4440   {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

4441     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4442     {

```

The command `\@@_exp_color_arg:NV` is *fully expandable*.

```

4443         \@@_exp_color_arg:NV \@@_rectanglecolor \l_tmpa_tl
4444         { \int_use:N \c@iRow - \int_use:N \c@jCol }
4445         { \int_use:N \c@iRow - * }
4446     }

```

Then, the other rows (if there is several rows).

```

4447     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4448     {
4449         \tl_gput_right:Nx \g_nicematrix_code_before_tl
4450         {
4451             \@@_exp_color_arg:NV \@@_rowcolor \l_tmpa_tl
4452             {
4453                 \int_eval:n { \c@iRow + 1 }
4454                 - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4455             }
4456         }
4457     }
4458 }
4459 \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4460 \tl_gput_right:Nx \g_@@_row_style_tl
4461 { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4462 \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.

```

4463     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4464     {
4465         \tl_gput_right:Nx \g_@@_row_style_tl
4466         {
4467             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4468             {
4469                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
4470                 { \dim_use:N \l_tmpa_dim }
4471             }
4472         }
4473     }

```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```

4474     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4475     {
4476         \tl_gput_right:Nx \g_@@_row_style_tl
4477         {
4478             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4479             {
4480                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4481                 { \dim_use:N \l_tmpb_dim }
4482             }
4483         }
4484     }

```

\l_@@_color_tl is the value of the key color of \RowStyle.

```

4485     \tl_if_empty:NF \l_@@_color_tl
4486     {
4487         \tl_gput_right:Nx \g_@@_row_style_tl
4488         {
4489             \mode_leave_vertical:
4490             \@@_color:n { \l_@@_color_tl }
4491         }
4492     }

```

\l_tmpa_bool is the value of the key bold.

```

4493     \bool_if:NT \l_tmpa_bool
4494     {
4495         \tl_gput_right:Nn \g_@@_row_style_tl
4496         {
4497             \if_mode_math:
4498             \c_math_toggle_token
4499             \bfseries \boldmath

```

```

4500         \c_math_toggle_token
4501     \else:
4502         \bfseries \boldmath
4503     \fi:
4504 }
4505 }
4506 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4507 \group_end:
4508 \g_@@_row_style_tl
4509 \ignorespaces
4510 }

```

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).

```

4511 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4512 {

```

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.

```

4513     \int_zero:N \l_tmpa_int
4514     \seq_map_indexed_inline:Nn \g_@@_colors_seq
4515     { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4516     \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

4517     {
4518         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4519         \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4520     }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

4521     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
4522 }

```

```

4523 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4524 \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`).

```

4525 \cs_new_protected:Npn \@@_actually_color:
4526 {

```

```

4527 \pgfpicture
4528 \pgf@relevantforpicturesizefalse
4529 \seq_map_indexed_inline:Nn \g_@@_colors_seq
4530 {
4531   \color ##2
4532   \use:c { g_@@_color _ ##1 _tl }
4533   \tl_gclear:c { g_@@_color _ ##1 _tl }
4534   \pgfusepath { fill }
4535 }
4536 \endpgfpicture
4537 }
4538 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4539 {
4540   \tl_set:Nn \l_@@_rows_tl { #1 }
4541   \tl_set:Nn \l_@@_cols_tl { #2 }
4542   \@@_cartesian_path:
4543 }

```

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

```

4544 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4545 {
4546   \tl_if_blank:nF { #2 }
4547   {
4548     \@@_add_to_colors_seq:xn
4549     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4550     { \@@_cartesian_color:nn { #3 } { - } }
4551   }
4552 }

```

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

```

4553 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4554 {
4555   \tl_if_blank:nF { #2 }
4556   {
4557     \@@_add_to_colors_seq:xn
4558     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4559     { \@@_cartesian_color:nn { - } { #3 } }
4560   }
4561 }

```

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

```

4562 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4563 {
4564   \tl_if_blank:nF { #2 }
4565   {
4566     \@@_add_to_colors_seq:xn
4567     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4568     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4569   }
4570 }

```

The last argument is the radius of the corners of the rectangle.

```

4571 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4572 {
4573   \tl_if_blank:nF { #2 }
4574   {
4575     \@@_add_to_colors_seq:xn
4576     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4577     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4578   }
4579 }

```

The last argument is the radius of the corners of the rectangle.

```

4580 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4581 {
4582   \@@_cut_on_hyphen:w #1 \q_stop
4583   \tl_clear_new:N \l_@@_tmpc_tl
4584   \tl_clear_new:N \l_@@_tmpd_tl
4585   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
4586   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
4587   \@@_cut_on_hyphen:w #2 \q_stop
4588   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
4589   \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`.

```

4590   \@@_cartesian_path:n { #3 }
4591 }

```

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

```

4592 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4593 {
4594   \clist_map_inline:nn { #3 }
4595   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4596 }

```

```

4597 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4598 {
4599   \int_step_inline:nn { \int_use:N \c@iRow }
4600   {
4601     \int_step_inline:nn { \int_use:N \c@jCol }
4602     {
4603       \int_if_even:nTF { ####1 + ##1 }
4604       { \@@_cellcolor [ #1 ] { #2 } }
4605       { \@@_cellcolor [ #1 ] { #3 } }
4606       { ##1 - ####1 }
4607     }
4608   }
4609 }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns) and the cells in the “corners”.

```

4610 \NewDocumentCommand \@@_arraycolor { 0 { } m }
4611 {
4612   \@@_rectanglecolor [ #1 ] { #2 }
4613   { 1 - 1 }
4614   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4615 }

4616 \keys_define:nn { NiceMatrix / rowcolors }
4617 {
4618   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4619   respect-blocks .default:n = true ,
4620   cols .tl_set:N = \l_@@_cols_tl ,
4621   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4622   restart .default:n = true ,
4623   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
4624 }

```

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*.

```
4625 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
4626 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```
4627 \group_begin:
4628 \seq_clear_new:N \l_@@_colors_seq
4629 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
4630 \tl_clear_new:N \l_@@_cols_tl
4631 \tl_set:Nn \l_@@_cols_tl { - }
4632 \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).

```
4633 \int_zero_new:N \l_@@_color_int
4634 \int_set:Nn \l_@@_color_int 1
4635 \bool_if:NT \l_@@_respect_blocks_bool
4636 {
```

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`).

```
4637 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4638 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
4639 { \@@_not_in_exterior_p:nnnnn #1 }
4640 }
4641 \pgfpicture
4642 \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
4643 \clist_map_inline:nn { #2 }
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 { \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.

```
4649 \int_set:Nn \l_tmpa_int \l_tmpa_tl
4650 \bool_if:NNTF \l_@@_rowcolors_restart_bool
4651 { \int_set:Nn \l_@@_color_int 1 }
4652 { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
4653 \int_zero_new:N \l_@@_tmpc_int
4654 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
4655 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
4656 {
```

We will compute in `\l_tmpb_int` the last row of the “block”.

```
4657 \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).

```
4658 \bool_if:NT \l_@@_respect_blocks_bool
4659 {
4660 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
4661 { \@@_intersect_our_row_p:nnnnn ###1 }
4662 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }
```


Now, the last row of the block is computed in `\l_tmpb_int`.

```

4663     }
4664     \tl_set:Nx \l_@@_rows_tl
4665     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
\l_@@_tmpc_tl will be the color that we will use.
4666     \tl_clear_new:N \l_@@_color_tl
4667     \tl_set:Nx \l_@@_color_tl
4668     {
4669         \@@_color_index:n
4670         {
4671             \int_mod:nn
4672             { \l_@@_color_int - 1 }
4673             { \seq_count:N \l_@@_colors_seq }
4674             + 1
4675         }
4676     }
4677     \tl_if_empty:NF \l_@@_color_tl
4678     {
4679         \@@_add_to_colors_seq:xx
4680         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
4681         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
4682     }
4683     \int_incr:N \l_@@_color_int
4684     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4685 }
4686 }
4687 \endpgfpicture
4688 \group_end:
4689 }

```

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.

```

4690 \cs_new:Npn \@@_color_index:n #1
4691 {
4692     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
4693     { \@@_color_index:n { #1 - 1 } }
4694     { \seq_item:Nn \l_@@_colors_seq { #1 } }
4695 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

4696 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4697 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

4698 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
4699 {
4700     \int_compare:nNnT { #3 } > \l_tmpb_int
4701     { \int_set:Nn \l_tmpb_int { #3 } }
4702 }

4703 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
4704 {
4705     \bool_lazy_or:nnTF
4706     { \int_compare_p:nNn { #4 } = \c_zero_int }
4707     { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
4708     \prg_return_false:
4709     \prg_return_true:
4710 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

4711 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
4712 {
4713   \bool_if:nTF
4714   {
4715     \int_compare_p:n { #1 <= \l_tmpa_int }
4716     &&
4717     \int_compare_p:n { \l_tmpa_int <= #3 }
4718   }
4719   \prg_return_true:
4720   \prg_return_false:
4721 }

```

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).

```

4722 \cs_new_protected:Npn \@@_cartesian_path:n #1
4723 {
4724   \bool_lazy_and:nnT
4725   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4726   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4727   {
4728     \@@_expand_clist:NN \l_@@_cols_tl \c_jCol
4729     \@@_expand_clist:NN \l_@@_rows_tl \c_iRow
4730   }

```

We begin the loop over the columns.

```

4731 \clist_map_inline:Nn \l_@@_cols_tl
4732 {
4733   \tl_set:Nn \l_tmpa_tl { ##1 }
4734   \tl_if_in:NnTF \l_tmpa_tl { - }
4735   { \@@_cut_on_hyphen:w ##1 \q_stop }
4736   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4737   \bool_lazy_or:nnT
4738   { \tl_if_blank_p:V \l_tmpa_tl }
4739   { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4740   { \tl_set:Nn \l_tmpa_tl { 1 } }
4741   \bool_lazy_or:nnT
4742   { \tl_if_blank_p:V \l_tmpb_tl }
4743   { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4744   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c_jCol } }
4745   \int_compare:nNnT \l_tmpb_tl > \c_jCol
4746   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c_jCol } }

```

\l_@@_tmpc_tl will contain the number of column.

```

4747 \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.

```

4748 \@@_qpoint:n { col - \l_tmpa_tl }
4749 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4750 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4751 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4752 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
4753 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4754 \clist_map_inline:Nn \l_@@_rows_tl
4755 {
4756   \tl_set:Nn \l_tmpa_tl { ####1 }

```

```

4757 \tl_if_in:NnTF \l_tmpa_tl { - }
4758 { \@@_cut_on_hyphen:w #####1 \q_stop }
4759 { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4760 \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4761 \tl_if_empty:NT \l_tmpb_tl
4762 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4763 \int_compare:nNnT \l_tmpb_tl > \c@iRow
4764 { \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`.

```

4765 \seq_if_in:NxF \l_@@_corners_cells_seq
4766 { \l_tmpa_tl - \l_@@_tmpc_tl }
4767 {
4768   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
4769   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4770   \@@_qpoint:n { row - \l_tmpa_tl }
4771   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4772   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4773   \pgfpathrectanglecorners
4774     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
4775     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4776 }
4777 }
4778 }
4779 }

```

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`).

```

4780 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to 2,4-6,8-* and `\c@jCol` equal to 10, the clist `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

4781 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4782 {
4783   \clist_set_eq:NN \l_tmpa_clist #1
4784   \clist_clear:N #1
4785   \clist_map_inline:Nn \l_tmpa_clist
4786   {
4787     \tl_set:Nn \l_tmpa_tl { ##1 }
4788     \tl_if_in:NnTF \l_tmpa_tl { - }
4789     { \@@_cut_on_hyphen:w ##1 \q_stop }
4790     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4791     \bool_lazy_or:nnT
4792     { \tl_if_blank_p:V \l_tmpa_tl }
4793     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4794     { \tl_set:Nn \l_tmpa_tl { 1 } }
4795     \bool_lazy_or:nnT
4796     { \tl_if_blank_p:V \l_tmpb_tl }
4797     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4798     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4799     \int_compare:nNnT \l_tmpb_tl > #2
4800     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4801     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4802     { \clist_put_right:Nn #1 { #####1 } }
4803   }
4804 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```

4805 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4806 {

```

```

4807 \peek_remove_spaces:n
4808 {
4809     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4810     {

```

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).

```

4811         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
4812         { \int_use:N \c@iRow - \int_use:N \c@jCol }
4813     }
4814 }
4815 }

```

When the user uses the key colortbl-like, the following command will be linked to \rowcolor in the tabular.

```

4816 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4817 {
4818     \peek_remove_spaces:n
4819     {
4820         \tl_gput_right:Nx \g_nicematrix_code_before_tl
4821         {
4822             \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4823             { \int_use:N \c@iRow - \int_use:N \c@jCol }
4824             { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4825         }
4826     }
4827 }

```

```

4828 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4829 {

```

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).

```

4830     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4831     {

```

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).

```

4832         \tl_gput_left:Nx \g_nicematrix_code_before_tl
4833         {
4834             \exp_not:N \columncolor [ #1 ]
4835             { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4836         }
4837     }
4838 }

```

The vertical and horizontal rules

OnlyMainNiceMatrix

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.

```

4839 \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`.

```

4840 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4841 {
4842   \int_compare:nNnTF \l_@@_first_col_int = 0
4843   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4844   {
4845     \int_compare:nNnTF \c@jCol = 0
4846     {
4847       \int_compare:nNnF \c@iRow = { -1 }
4848       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4849     }
4850     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4851   }
4852 }

```

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.

```

4853 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4854 {
4855   \int_compare:nNnF \c@iRow = 0
4856   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4857 }

```

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`).

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of `key=value` pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```

4858 \keys_define:nn { NiceMatrix / Rules }
4859 {
4860   position .int_set:N = \l_@@_position_int ,
4861   position .value_required:n = true ,
4862   start .int_set:N = \l_@@_start_int ,
4863   start .initial:n = 1 ,
4864   end .code:n =
4865     \bool_lazy_or:nnTF
4866     { \tl_if_empty_p:n { #1 } }
4867     { \str_if_eq_p:nn { #1 } { last } }
4868     { \int_set_eq:NN \l_@@_end_int \c@jCol }
4869     { \int_set:Nn \l_@@_end_int { #1 } }
4870 }

```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

4871 \keys_define:nn { NiceMatrix / RulesBis }
4872 {
4873   multiplicity .int_set:N = \l_@@_multiplicity_int ,
4874   multiplicity .initial:n = 1 ,
4875   dotted .bool_set:N = \l_@@_dotted_bool ,
4876   dotted .initial:n = false ,

```

```

4877 dotted .default:n = true ,
4878 color .code:n = \@@_set_CT@arc@:n { #1 } ,
4879 color .value_required:n = true ,
4880 sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
4881 sep-color .value_required:n = true ,

```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with Tikz.

```

4882 tikz .tl_set:N = \l_@@_tikz_rule_tl ,
4883 tikz .value_required:n = true ,
4884 tikz .initial:n = ,
4885 total-width .dim_set:N = \l_@@_rule_width_dim ,
4886 total-width .value_required:n = true ,
4887 width .meta:n = { total-width = #1 }
4888 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

4889 \cs_new_protected:Npn \@@_vline:n #1
4890 {

```

The group is for the options.

```

4891 \group_begin:
4892 \int_zero_new:N \l_@@_end_int
4893 \int_set_eq:NN \l_@@_end_int \c@iRow
4894 \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl

```

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).

```

4895 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
4896 \@@_vline_i:
4897 \group_end:
4898 }

```

```

4899 \cs_new_protected:Npn \@@_vline_i:
4900 {
4901 \int_zero_new:N \l_@@_local_start_int
4902 \int_zero_new:N \l_@@_local_end_int

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

4903 \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
4904 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
4905 \l_tmpa_tl
4906 {

```

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.

```

4907 \bool_gset_true:N \g_tmpa_bool
4908 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4909 { \@@_test_vline_in_block:nnnnn #1 }
4910 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4911 { \@@_test_vline_in_block:nnnnn #1 }
4912 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4913 { \@@_test_vline_in_stroken_block:nnnn #1 }
4914 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
4915 \bool_if:NTF \g_tmpa_bool
4916 {
4917 \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. \l_@@_local_start_int will be the starting row of the rule that we will have to draw.

```

4918         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
4919     }
4920     {
4921         \int_compare:nNnT \l_@@_local_start_int > 0
4922         {
4923             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
4924             \@@_vline_ii:
4925             \int_zero:N \l_@@_local_start_int
4926         }
4927     }
4928 }
4929 \int_compare:nNnT \l_@@_local_start_int > 0
4930 {
4931     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
4932     \@@_vline_ii:
4933 }
4934 }

4935 \cs_new_protected:Npn \@@_test_in_corner_v:
4936 {
4937     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
4938     {
4939         \seq_if_in:NxT
4940         \l_@@_corners_cells_seq
4941         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
4942         { \bool_set_false:N \g_tmpa_bool }
4943     }
4944     {
4945         \seq_if_in:NxT
4946         \l_@@_corners_cells_seq
4947         { \l_tmpa_tl - \l_tmpb_tl }
4948         {
4949             \int_compare:nNnTF \l_tmpb_tl = 1
4950             { \bool_set_false:N \g_tmpa_bool }
4951             {
4952                 \seq_if_in:NxT
4953                 \l_@@_corners_cells_seq
4954                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
4955                 { \bool_set_false:N \g_tmpa_bool }
4956             }
4957         }
4958     }
4959 }

4960 \cs_new_protected:Npn \@@_vline_ii:
4961 {
4962     \bool_set_false:N \l_@@_dotted_bool
4963     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
4964     \bool_if:NTF \l_@@_dotted_bool
4965     \@@_vline_iv:
4966     {
4967         \tl_if_empty:NTF \l_@@_tikz_rule_tl
4968         \@@_vline_iii:
4969         \@@_vline_v:
4970     }
4971 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

4972 \cs_new_protected:Npn \@@_vline_iii:

```

```

4973 {
4974   \pgfpicture
4975   \pgfrememberpicturepositiononpagetrue
4976   \pgf@relevantforpicturesizefalse
4977   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4978   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4979   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4980   \dim_set:Nn \l_tmpb_dim
4981     {
4982       \pgf@x
4983       - 0.5 \l_@@_rule_width_dim
4984       +
4985       ( \arrayrulewidth * \l_@@_multiplicity_int
4986         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
4987     }
4988   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4989   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4990   \bool_lazy_all:nT
4991     {
4992       { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
4993       { \cs_if_exist_p:N \CT@drsc@ }
4994       { ! \tl_if_blank_p:V \CT@drsc@ }
4995     }
4996     {
4997       \group_begin:
4998       \CT@drsc@
4999       \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5000       \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5001       \dim_set:Nn \l_@@_tmpd_dim
5002         {
5003           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5004           * ( \l_@@_multiplicity_int - 1 )
5005         }
5006       \pgfpathrectanglecorners
5007         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5008         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5009       \pgfusepath { fill }
5010       \group_end:
5011     }
5012   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5013   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5014   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5015     {
5016       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5017       \dim_sub:Nn \l_tmpb_dim \doublerulesep
5018       \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5019       \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5020     }
5021   \CT@arc@
5022   \pgfsetlinewidth { 1.1 \arrayrulewidth }
5023   \pgfsetrectcap
5024   \pgfusepathqstroke
5025   \endpgfpicture
5026 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

5027 \cs_new_protected:Npn \@@_vline_iv:
5028 {
5029   \pgfpicture
5030   \pgfrememberpicturepositiononpagetrue
5031   \pgf@relevantforpicturesizefalse
5032   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5033   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }

```



```

5034 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5035 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5036 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5037 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5038 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5039 \CT@arc@
5040 \@@_draw_line:
5041 \endpgfpicture
5042 }

```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```

5043 \cs_new_protected:Npn \@@_vline_v:
5044 {
5045   \begin {tikzpicture }
5046   \pgfrememberpicturepositiononpagetrue
5047   \pgf@relevantforpicturesizefalse
5048   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5049   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5050   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5051   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5052   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5053   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5054   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5055   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5056   ( \l_tmpb_dim , \l_tmpa_dim ) --
5057   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5058   \end { tikzpicture }
5059 }

```

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).

```

5060 \cs_new_protected:Npn \@@_draw_vlines:
5061 {
5062   \int_step_inline:nnn
5063   {
5064     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5065     1 2
5066   }
5067   {
5068     \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5069     { \int_eval:n { \c@jCol + 1 } }
5070     \c@jCol
5071   }
5072   {
5073     \tl_if_eq:NnF \l_@@_vlines_clist { all }
5074     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5075     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5076   }
5077 }

```

The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs of the form `{NiceMatrix/Rules}`.

```

5078 \cs_new_protected:Npn \@@_hline:n #1
5079 {

```

The group is for the options.

```

5080   \group_begin:
5081   \int_zero_new:N \l_@@_end_int

```

```

5082 \int_set_eq:NN \l_@@_end_int \c@jCol
5083 \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5084 \@@_hline_i:
5085 \group_end:
5086 }
5087 \cs_new_protected:Npn \@@_hline_i:
5088 {
5089 \int_zero_new:N \l_@@_local_start_int
5090 \int_zero_new:N \l_@@_local_end_int

```

\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.

```

5091 \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5092 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5093 \l_tmpb_tl
5094 {

```

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.

```

5095 \bool_gset_true:N \g_tmpa_bool
5096 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5097 { \@@_test_hline_in_block:nnnnn ##1 }
5098 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5099 { \@@_test_hline_in_block:nnnnn ##1 }
5100 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5101 { \@@_test_hline_in_stroken_block:nnnn ##1 }
5102 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5103 \bool_if:NTF \g_tmpa_bool
5104 {
5105 \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. \l_@@_local_start_int will be the starting row of the rule that we will have to draw.

```

5106 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5107 }
5108 {
5109 \int_compare:nNnT \l_@@_local_start_int > 0
5110 {
5111 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5112 \@@_hline_ii:
5113 \int_zero:N \l_@@_local_start_int
5114 }
5115 }
5116 }
5117 \int_compare:nNnT \l_@@_local_start_int > 0
5118 {
5119 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5120 \@@_hline_ii:
5121 }
5122 }

```

```

5123 \cs_new_protected:Npn \@@_test_in_corner_h:
5124 {
5125 \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5126 {
5127 \seq_if_in:NxT
5128 \l_@@_corners_cells_seq
5129 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5130 { \bool_set_false:N \g_tmpa_bool }
5131 }
5132 {

```

```

5133 \seq_if_in:NxT
5134 \l_@@_corners_cells_seq
5135 { \l_tmpa_tl - \l_tmpb_tl }
5136 {
5137   \int_compare:nNnTF \l_tmpa_tl = 1
5138   { \bool_set_false:N \g_tmpa_bool }
5139   {
5140     \seq_if_in:NxT
5141     \l_@@_corners_cells_seq
5142     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5143     { \bool_set_false:N \g_tmpa_bool }
5144   }
5145 }
5146 }
5147 }

```

```

5148 \cs_new_protected:Npn \@@_hline_ii:
5149 {
5150   \bool_set_false:N \l_@@_dotted_bool
5151   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5152   \bool_if:NTF \l_@@_dotted_bool
5153   \@@_hline_iv:
5154   {
5155     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5156     \@@_hline_iii:
5157     \@@_hline_v:
5158   }
5159 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

5160 \cs_new_protected:Npn \@@_hline_iii:
5161 {
5162   \pgfpicture
5163   \pgfrememberpicturepositiononpagetrue
5164   \pgf@relevantforpicturesizefalse
5165   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5166   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5167   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5168   \dim_set:Nn \l_tmpb_dim
5169   {
5170     \pgf@y
5171     - 0.5 \l_@@_rule_width_dim
5172     +
5173     ( \arrayrulewidth * \l_@@_multiplicity_int
5174       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5175   }
5176   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5177   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5178   \bool_lazy_all:nT
5179   {
5180     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5181     { \cs_if_exist_p:N \CT@drsc@ }
5182     { ! \tl_if_blank_p:V \CT@drsc@ }
5183   }
5184   {
5185     \group_begin:
5186     \CT@drsc@
5187     \dim_set:Nn \l_@@_tmpd_dim
5188     {
5189       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5190       * ( \l_@@_multiplicity_int - 1 )
5191     }

```

```

5192     \pgfpathrectanglecorners
5193     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5194     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5195     \pgfusepathqfill
5196     \group_end:
5197 }
5198 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5199 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5200 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5201 {
5202     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5203     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5204     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5205     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5206 }
5207 \CT@arc@
5208 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5209 \pgfsetrectcap
5210 \pgfusepathqstroke
5211 \endpgfpicture
5212 }

```

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}

```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

```

5213 \cs_new_protected:Npn \l_@@_hline_iv:
5214 {
5215     \pgfpicture
5216     \pgfrememberpicturepositiononpagetrue
5217     \pgf@relevantforpicturesizefalse
5218     \l_@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5219     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5220     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
5221     \l_@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5222     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5223     \int_compare:nNnT \l_@@_local_start_int = 1
5224     {
5225         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5226         \bool_if:NT \g_@@_NiceArray_bool
5227         { \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_@@_xdots_inter_dim` is *ad hoc* for a better result.

```

5228     \tl_if_eq:NnF \g_@@_left_delim_tl (
5229     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
5230     )
5231     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5232     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x

```

```

5233 \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5234 {
5235     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5236     \bool_if:NT \g_@@_NiceArray_bool
5237     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5238     \tl_if_eq:NnF \g_@@_right_delim_tl )
5239     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
5240 }
5241 \CT@arc@
5242 \@@_draw_line:
5243 \endpgfpicture
5244 }

```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```

5245 \cs_new_protected:Npn \@@_hline_v:
5246 {
5247     \begin { tikzpicture }
5248     \pgfrememberpicturepositiononpagetrue
5249     \pgf@relevantforpicturesizefalse
5250     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5251     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5252     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5253     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5254     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5255     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5256     \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5257     \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5258     ( \l_tmpa_dim , \l_tmpb_dim ) --
5259     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5260     \end { tikzpicture }
5261 }

```

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)).

```

5262 \cs_new_protected:Npn \@@_draw_hlines:
5263 {
5264     \int_step_inline:nnn
5265     {
5266         \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5267         1 2
5268     }
5269     {
5270         \bool_if:nTF { \g_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5271         { \int_eval:n { \c@iRow + 1 } }
5272         \c@iRow
5273     }
5274     {
5275         \tl_if_eq:NnF \l_@@_hlines_clist { all }
5276         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5277         { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
5278     }
5279 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5280 \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.

```

5281 \cs_set:Npn \@@_Hline_i:n #1
5282 {
5283     \peek_remove_spaces:n

```

```

5284     {
5285       \peek_meaning:NTF \Hline
5286       { \@@_Hline_ii:nn { #1 + 1 } }
5287       { \@@_Hline_iii:n { #1 } }
5288     }
5289   }
5290 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5291 \cs_set:Npn \@@_Hline_iii:n #1
5292   {
5293     \skip_vertical:n
5294     {
5295       \arrayrulewidth * ( #1 )
5296       + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
5297     }
5298     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5299     {
5300       \@@_hline:n
5301       {
5302         position = \int_eval:n { \c@iRow + 1 } ,
5303         multiplicity = #1
5304       }
5305     }
5306     \ifnum 0 = `{ \fi }
5307   }

```

Customized rules defined by the final user

The final user can define a customized rule by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs.

Among the keys available in that list, there is the key `letter` to specify a letter that the final user will use in the preamble of the array. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix / ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```

5308 \keys_define:nn { NiceMatrix / ColumnTypes } { }

```

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```

5309 \cs_new_protected:Npn \@@_custom_line:n #1
5310   {
5311     \str_clear_new:N \l_@@_command_str
5312     \str_clear_new:N \l_@@_ccommand_str
5313     \str_clear_new:N \l_@@_letter_str
5314     \keys_set_known:nnN { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

5315   \bool_lazy_all:nTF
5316     {
5317       { \str_if_empty_p:N \l_@@_letter_str }
5318       { \str_if_empty_p:N \l_@@_command_str }
5319       { \str_if_empty_p:N \l_@@_ccommand_str }
5320     }
5321     { \@@_error:n { No-letter~and~no-command } }
5322     { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
5323   }

```

```

5324 \keys_define:nn { NiceMatrix / custom-line }
5325 {
5326   % here, we will use change in the future to use .str_set:N
5327   letter .code:n = \str_set:Nn \l_@@_letter_str { #1 } ,
5328   letter .value_required:n = true ,
5329   command .code:n = \str_set:Nn \l_@@_command_str { #1 } ,
5330   command .value_required:n = true ,
5331   ccommand .code:n = \str_set:Nn \l_@@_ccommand_str { #1 } ,
5332   ccommand .value_required:n = true ,
5333 }

```

```

5334 \cs_new_protected:Npn \@@_custom_line_i:n #1
5335 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

5336   \bool_set_false:N \l_@@_tikz_rule_bool
5337   \bool_set_false:N \l_@@_dotted_rule_bool
5338   \bool_set_false:N \l_@@_color_bool
5339   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
5340   \bool_if:NT \l_@@_tikz_rule_bool
5341   {

```

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.

```

5342     \cs_if_exist:NF \tikzpicture
5343     { \@@_error:n { tikz-in-custom-line-without-tikz } }
5344     \bool_if:NT \l_@@_color_bool
5345     { \@@_error:n { color-in-custom-line-with-tikz } }
5346   }
5347   \bool_if:nT
5348   {
5349     \int_compare_p:nNn \l_@@_multiplicity_int > 1
5350     && \l_@@_dotted_rule_bool
5351   }
5352   { \@@_error:n { key-multiplicity-with-dotted } }
5353   \str_if_empty:NF \l_@@_letter_str
5354   {
5355     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
5356     { \@@_error:n { Several-letters } }
5357     {
5358       \exp_args:NnV \tl_if_in:NnTF
5359       \c_@@_forbidden_letters_str \l_@@_letter_str
5360       { \@@_error:n { Forbidden-letter } }
5361       {

```

The final user can, locally, redefine a letter of column type. That's compatible with the use of `\keys_define:nn`: the definition is local and may overwrite a previous definition.

```

5362         \keys_define:nx { NiceMatrix / ColumnTypes }
5363         {
5364           \l_@@_letter_str .code:n =
5365           { \@@_v_custom_line:n { \exp_not:n { #1 } } }
5366         }
5367       }
5368     }
5369   }
5370   \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
5371   \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
5372 }
5373 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

5374 \keys_define:nn { NiceMatrix / custom-line-bis }
5375 {
5376   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5377   multiplicity .initial:n = 1 ,
5378   multiplicity .value_required:n = true ,
5379   color .code:n = \bool_set_true:N \l_@@_color_bool ,
5380   color .value_required:n = true ,
5381   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5382   tikz .value_required:n = true ,
5383   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5384   dotted .value_forbidden:n = true ,
5385   total-width .code:n = { } ,
5386   total-width .value_required:n = true ,
5387   width .code:n = { } ,
5388   width .value_required:n = true ,
5389   sep-color .code:n = { } ,
5390   sep-color .value_required:n = true ,
5391   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
5392 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

5393 \bool_new:N \l_@@_dotted_rule_bool
5394 \bool_new:N \l_@@_tikz_rule_bool
5395 \bool_new:N \l_@@_color_bool

```

The following keys are used to determine the total width of the line (including the spaces on both sides of the line). The key `width` is deprecated and has been replaced by the key `total-width`.

```

5396 \keys_define:nn { NiceMatrix / custom-line-width }
5397 {
5398   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5399   multiplicity .initial:n = 1 ,
5400   multiplicity .value_required:n = true ,
5401   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5402   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5403                       \bool_set_true:N \l_@@_total_width_bool ,
5404   total-width .value_required:n = true ,
5405   width .meta:n = { total-width = #1 } ,
5406   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5407 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the 'h' in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

5408 \cs_new_protected:Npn \@@_h_custom_line:n #1
5409 {

```

We use `\cs_set:cpn` and not `\cs_new:cpn` because we want a local definition. Moreover, the command must *not* be protected since it begins with `\noalign`.

```

5410   \cs_set:cpn { nicematrix - \l_@@_command_str }
5411   {
5412     \noalign
5413     {
5414       \@@_compute_rule_width:n { #1 }
5415       \skip_vertical:n { \l_@@_rule_width_dim }
5416       \tl_gput_right:Nx \g_@@_internal_code_after_tl
5417       {
5418         \@@_hline:n

```



```

5419         {
5420             #1 ,
5421             position = \int_eval:n { \c@iRow + 1 } ,
5422             total-width = \dim_use:N \l_@@_rule_width_dim
5423         }
5424     }
5425 }
5426 }
5427 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
5428 }
5429 \cs_generate_variant:Nn \@@_h_custom_line:nn { n V }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter c as in `\cline`). #1 is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

5430 \cs_new_protected:Npn \@@_c_custom_line:n #1
5431 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

5432     \exp_args:Nc \NewExpandableDocumentCommand
5433     { nicematrix - \l_@@_ccommand_str }
5434     { 0 { } m }
5435     {
5436         \noalign
5437         {
5438             \@@_compute_rule_width:n { #1 , ##1 }
5439             \skip_vertical:n { \l_@@_rule_width_dim }
5440             \clist_map_inline:nn
5441             { ##2 }
5442             { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
5443         }
5444     }
5445     \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
5446 }

```

The first argument is the list of key-value pairs characteristic of the line. The second argument is the specification of columns for the `\cline` with the syntax *a-b*.

```

5447 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
5448 {
5449     \str_if_in:nnTF { #2 } { - }
5450     { \@@_cut_on_hyphen:w #2 \q_stop }
5451     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
5452     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5453     {
5454         \@@_hline:n
5455         {
5456             #1 ,
5457             start = \l_tmpa_tl ,
5458             end = \l_tmpb_tl ,
5459             position = \int_eval:n { \c@iRow + 1 } ,
5460             total-width = \dim_use:N \l_@@_rule_width_dim
5461         }
5462     }
5463 }
5464 \cs_generate_variant:Nn \@@_c_custom_line:nn { n V }
5465 \cs_new_protected:Npn \@@_compute_rule_width:n #1
5466 {
5467     \bool_set_false:N \l_@@_tikz_rule_bool
5468     \bool_set_false:N \l_@@_total_width_bool
5469     \bool_set_false:N \l_@@_dotted_rule_bool
5470     \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
5471     \bool_if:NF \l_@@_total_width_bool

```

```

5472 {
5473   \bool_if:NTF \l_@@_dotted_rule_bool
5474   { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
5475   {
5476     \bool_if:NF \l_@@_tikz_rule_bool
5477     {
5478       \dim_set:Nn \l_@@_rule_width_dim
5479       {
5480         \arrayrulewidth * \l_@@_multiplicity_int
5481         + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
5482       }
5483     }
5484   }
5485 }
5486 }

5487 \cs_new_protected:Npn \@@_v_custom_line:n #1
5488 {
5489   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

5490   \tl_gput_right:Nx \g_@@_preamble_tl
5491   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
5492   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5493   {
5494     \@@_vline:n
5495     {
5496       #1 ,
5497       position = \int_eval:n { \c@jCol + 1 } ,
5498       total-width = \dim_use:N \l_@@_rule_width_dim
5499     }
5500   }
5501 }

5502 \@@_custom_line:n
5503 { letter = : , command = hdottedline , ccommand = cdottedline, dotted }

```

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.

```

5504 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
5505 {
5506   \bool_lazy_all:nT
5507   {
5508     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
5509     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5510     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5511     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5512   }
5513   { \bool_gset_false:N \g_tmpa_bool }
5514 }

```

The same for vertical rules.

```

5515 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
5516 {
5517   \bool_lazy_all:nT
5518   {
5519     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5520     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5521     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
5522     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5523   }

```

```

5524     { \bool_gset_false:N \g_tmpa_bool }
5525   }
5526 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
5527 {
5528   \bool_lazy_all:nT
5529   {
5530     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5531     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
5532     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5533     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5534   }
5535   { \bool_gset_false:N \g_tmpa_bool }
5536 }
5537 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
5538 {
5539   \bool_lazy_all:nT
5540   {
5541     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5542     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5543     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5544     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
5545   }
5546   { \bool_gset_false:N \g_tmpa_bool }
5547 }

```

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.

```

5548 \cs_new_protected:Npn \@@_compute_corners:
5549 {

```

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.

```

5550   \seq_clear_new:N \l_@@_corners_cells_seq
5551   \clist_map_inline:Nn \l_@@_corners_clist
5552   {
5553     \str_case:nnF { ##1 }
5554     {
5555       { NW }
5556       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
5557       { NE }
5558       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
5559       { SW }
5560       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
5561       { SE }
5562       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
5563     }
5564     { \@@_error:nn { bad~corner } { ##1 } }
5565   }

```

Even if the user has used the `key corners` the list of cells in the corners may be empty.

```

5566   \seq_if_empty:NF \l_@@_corners_cells_seq
5567   {

```

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.

```

5568     \tl_gput_right:Nx \g_@@_aux_tl
5569     {
5570       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq

```

```

5571         { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
5572     }
5573 }
5574 }

```

“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.

```

5575 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
5576 {

```

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.

```

5577     \bool_set_false:N \l_tmpa_bool
5578     \int_zero_new:N \l_@@_last_empty_row_int
5579     \int_set:Nn \l_@@_last_empty_row_int { #1 }
5580     \int_step_inline:nnnn { #1 } { #3 } { #5 }
5581     {
5582         \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
5583         \bool_lazy_or:nnTF
5584         {
5585             \cs_if_exist_p:c
5586             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
5587         }
5588         \l_tmpb_bool
5589         { \bool_set_true:N \l_tmpa_bool }
5590         {
5591             \bool_if:NF \l_tmpa_bool
5592             { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
5593         }
5594     }

```

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

```

5595     \bool_set_false:N \l_tmpa_bool
5596     \int_zero_new:N \l_@@_last_empty_column_int
5597     \int_set:Nn \l_@@_last_empty_column_int { #2 }
5598     \int_step_inline:nnnn { #2 } { #4 } { #6 }
5599     {
5600         \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
5601         \bool_lazy_or:nnTF
5602         \l_tmpb_bool
5603         {
5604             \cs_if_exist_p:c
5605             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
5606         }
5607         { \bool_set_true:N \l_tmpa_bool }
5608         {
5609             \bool_if:NF \l_tmpa_bool
5610             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
5611         }
5612     }

```

Now, we loop over the rows.

```
5613 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
5614 {
```

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

```
5615 \bool_set_false:N \l_tmpa_bool
5616 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
5617 {
5618 \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
5619 \bool_lazy_or:nnTF
5620 \l_tmpb_bool
5621 {
5622 \cs_if_exist_p:c
5623 { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
5624 }
5625 { \bool_set_true:N \l_tmpa_bool }
5626 {
5627 \bool_if:NF \l_tmpa_bool
5628 {
5629 \int_set:Nn \l_@@_last_empty_column_int { ####1 }
5630 \seq_put_right:Nn
5631 \l_@@_corners_cells_seq
5632 { ##1 - ####1 }
5633 }
5634 }
5635 }
5636 }
5637 }
```

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`).

```
5638 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
5639 {
5640 \int_set:Nn \l_tmpa_int { #1 }
5641 \int_set:Nn \l_tmpb_int { #2 }
5642 \bool_set_false:N \l_tmpb_bool
5643 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5644 { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
5645 }
5646 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6 #7
5647 {
5648 \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
5649 {
5650 \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
5651 {
5652 \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
5653 {
5654 \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
5655 { \bool_set_true:N \l_tmpb_bool }
5656 }
5657 }
5658 }
5659 }
```

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.

```
5660 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

5661 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5662 {
5663   auto-columns-width .code:n =
5664   {
5665     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5666     \dim_gzero_new:N \g_@@_max_cell_width_dim
5667     \bool_set_true:N \l_@@_auto_columns_width_bool
5668   }
5669 }

5670 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
5671 {
5672   \int_gincr:N \g_@@_NiceMatrixBlock_int
5673   \dim_zero:N \l_@@_columns_width_dim
5674   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5675   \bool_if:NT \l_@@_block_auto_columns_width_bool
5676   {
5677     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5678     {
5679       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
5680       { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
5681     }
5682   }
5683 }

```

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`).

```

5684 {
5685   \bool_if:NT \l_@@_block_auto_columns_width_bool
5686   {
5687     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5688     \iow_shipout:Nx \@mainaux
5689     {
5690       \cs_gset:cpn
5691       { @@_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.

```

5692       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5693     }
5694     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5695   }
5696 }

```

The extra nodes

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

```

5697 \cs_generate_variant:Nn \dim_min:nn { v n }
5698 \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).

```

5699 \cs_new_protected:Npn \@@_create_extra_nodes:
5700 {
5701   \bool_if:nTF \l_@@_medium_nodes_bool
5702   {
5703     \bool_if:nTF \l_@@_large_nodes_bool
5704     \@@_create_medium_and_large_nodes:

```

```

5705     \@@_create_medium_nodes:
5706   }
5707   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5708 }

```

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`.

```

5709 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5710 {
5711   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5712   {
5713     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
5714     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
5715     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
5716     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
5717   }
5718   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5719   {
5720     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
5721     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
5722     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
5723     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
5724   }

```

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

```

5725   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5726   {
5727     \int_step_variable:nnNn
5728     \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.

```

5729   {
5730     \cs_if_exist:cT
5731     { 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`.

```

5732   {
5733     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5734     \dim_set:cn { l_@@_row\_@@_i: _min_dim }
5735     { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
5736     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5737     {
5738       \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
5739       { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
5740     }

```

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`.

```

5741         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
5742         \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
5743         { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
5744         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5745         {
5746             \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
5747             { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
5748         }
5749     }
5750 }
5751 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

5752 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5753 {
5754     \dim_compare:nNnT
5755     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
5756     {
5757         \@@_qpoint:n { row - \@@_i: - base }
5758         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
5759         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
5760     }
5761 }
5762 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5763 {
5764     \dim_compare:nNnT
5765     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
5766     {
5767         \@@_qpoint:n { col - \@@_j: }
5768         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
5769         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
5770     }
5771 }
5772 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

5773 \cs_new_protected:Npn \@@_create_medium_nodes:
5774 {
5775     \pgfpicture
5776     \pgfrememberpicturepositiononpagetrue
5777     \pgf@relevantforpicturesizefalse
5778     \@@_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”.

```

5779         \tl_set:Nn \l_@@_suffix_tl { -medium }
5780         \@@_create_nodes:
5781         \endpgfpicture
5782     }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁷³. 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:`.

⁷³If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`


```

5783 \cs_new_protected:Npn \@@_create_large_nodes:
5784 {
5785   \pgfpicture
5786     \pgfrememberpicturepositiononpagetrue
5787     \pgf@relevantforpicturesizefalse
5788     \@@_computations_for_medium_nodes:
5789     \@@_computations_for_large_nodes:
5790     \tl_set:Nn \l_@@_suffix_tl { - large }
5791     \@@_create_nodes:
5792   \endpgfpicture
5793 }
5794 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5795 {
5796   \pgfpicture
5797     \pgfrememberpicturepositiononpagetrue
5798     \pgf@relevantforpicturesizefalse
5799     \@@_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”.

```

5800   \tl_set:Nn \l_@@_suffix_tl { - medium }
5801   \@@_create_nodes:
5802   \@@_computations_for_large_nodes:
5803   \tl_set:Nn \l_@@_suffix_tl { - large }
5804   \@@_create_nodes:
5805   \endpgfpicture
5806 }

```

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.

```

5807 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5808 {
5809   \int_set:Nn \l_@@_first_row_int 1
5810   \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`.

```

5811   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5812   {
5813     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5814     {
5815       (
5816         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5817         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
5818       )
5819       / 2
5820     }
5821     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
5822     { l_@@_row _ \@@_i: _ min_dim }
5823   }
5824   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5825   {
5826     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5827     {
5828       (
5829         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5830         \dim_use:c
5831         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
5832       )
5833       / 2
5834     }
5835     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
5836     { l_@@_column _ \@@_j: _ max _ dim }
5837   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

5838   \dim_sub:cn
5839   { l_@@_column _ 1 _ min _ dim }
5840   \l_@@_left_margin_dim
5841   \dim_add:cn
5842   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5843   \l_@@_right_margin_dim
5844 }

```

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).

```

5845 \cs_new_protected:Npn \@@_create_nodes:
5846 {
5847   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5848   {
5849     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5850     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

5851       \@@_pgf_rect_node:nnnnn
5852       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5853       { \dim_use:c { l_@@_column _ \@@_j: _min_dim } }
5854       { \dim_use:c { l_@@_row _ \@@_i: _min_dim } }
5855       { \dim_use:c { l_@@_column _ \@@_j: _max_dim } }
5856       { \dim_use:c { l_@@_row _ \@@_i: _max_dim } }
5857       \str_if_empty:NF \l_@@_name_str
5858       {
5859         \pgfnodealias
5860         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5861         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5862       }
5863     }
5864   }

```

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 .

```

5865   \seq_mapthread_function:NNN
5866   \g_@@_multicolumn_cells_seq
5867   \g_@@_multicolumn_sizes_seq
5868   \@@_node_for_multicolumn:nn
5869 }

```

```

5870 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5871 {
5872   \cs_set_nopar:Npn \@@_i: { #1 }
5873   \cs_set_nopar:Npn \@@_j: { #2 }
5874 }

```

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”).

```

5875 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5876 {
5877   \@@_extract_coords_values: #1 \q_stop
5878   \@@_pgf_rect_node:nnnnn
5879   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5880   { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } }

```

```

5881 { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
5882 { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
5883 { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
5884 \str_if_empty:NF \l_@@_name_str
5885 {
5886   \pgfnodealias
5887   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5888   { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
5889 }
5890 }

```

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.

```

5891 \keys_define:nn { NiceMatrix / Block / FirstPass }
5892 {
5893   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5894   l .value_forbidden:n = true ,
5895   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5896   r .value_forbidden:n = true ,
5897   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5898   c .value_forbidden:n = true ,
5899   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5900   L .value_forbidden:n = true ,
5901   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5902   R .value_forbidden:n = true ,
5903   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5904   C .value_forbidden:n = true ,
5905   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5906   t .value_forbidden:n = true ,
5907   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5908   b .value_forbidden:n = true ,
5909   color .tl_set:N = \l_@@_color_tl ,
5910   color .value_required:n = true ,
5911   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
5912   respect-arraystretch .default:n = true ,
5913 }

```

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.

```

5914 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
5915 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not been provided by the user, you use 1-1 (that is to say a block of only one cell).

```

5916 \peek_remove_spaces:n
5917 {
5918   \tl_if_blank:nTF { #2 }
5919   { \@@_Block_i 1-1 \q_stop }
5920   { \@@_Block_i #2 \q_stop }
5921   { #1 } { #3 } { #4 }
5922 }
5923 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5924 \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 $key=value$ pairs, #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.

```
5925 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5926 {
```

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).

```
5927   \bool_lazy_or:nnTF
5928     { \tl_if_blank_p:n { #1 } }
5929     { \str_if_eq_p:nn { #1 } { * } }
5930     { \int_set:Nn \l_tmpa_int { 100 } }
5931     { \int_set:Nn \l_tmpa_int { #1 } }
5932   \bool_lazy_or:nnTF
5933     { \tl_if_blank_p:n { #2 } }
5934     { \str_if_eq_p:nn { #2 } { * } }
5935     { \int_set:Nn \l_tmpb_int { 100 } }
5936     { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
5937   \int_compare:nNnTF \l_tmpb_int = 1
5938   {
5939     \str_if_empty:NTF \l_@@_hpos_cell_str
5940     { \str_set:Nn \l_@@_hpos_block_str c }
5941     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
5942   }
5943   { \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.

```
5944   \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
5945   \tl_set:Nx \l_tmpa_tl
5946   {
5947     { \int_use:N \c@iRow }
5948     { \int_use:N \c@jCol }
5949     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5950     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5951   }
```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

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

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by curryfication).

```
5952   \bool_if:nTF
5953   {
5954     (
5955       \int_compare_p:nNn { \l_tmpa_int } = 1
5956       ||
5957       \int_compare_p:nNn { \l_tmpb_int } = 1
5958     )
5959     && ! \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.

```

5960      && ! \l_@@_X_column_bool
5961    }
5962    { \exp_args:Nxx \@@_Block_iv:nnnnn }
5963    { \exp_args:Nxx \@@_Block_v:nnnnn }
5964    { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5965  }

```

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).

```

5966 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5967 {
5968   \int_gincr:N \g_@@_block_box_int
5969   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5970   {
5971     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5972     {
5973       \@@_actually_diagbox:nnnnnn
5974       { \int_use:N \c@iRow }
5975       { \int_use:N \c@jCol }
5976       { \int_eval:n { \c@iRow + #1 - 1 } }
5977       { \int_eval:n { \c@jCol + #2 - 1 } }
5978       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5979     }
5980   }
5981   \box_gclear_new:c
5982   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5983   \hbox_gset:cn
5984   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5985   {

```

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}).

```

5986     \tl_if_empty:NTF \l_@@_color_tl
5987     { \int_compare:nNnT { #2 } = 1 \set@color }
5988     { \@@_color:V \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.

```

5989     \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5990     \group_begin:
5991     \bool_if:NF \l_@@_respect_arraystretch_bool
5992     { \cs_set:Npn \arraystretch { 1 } }
5993     \dim_zero:N \extrarowheight
5994     #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).

```

5995     \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
5996     \bool_if:NTF \l_@@_NiceTabular_bool
5997     {
5998       \bool_lazy_all:nTF
5999       {

```

```

6000         { \int_compare_p:nNn { #2 } = 1 }
6001         { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6002         { ! \l_@@_respect_arraystretch_bool }
6003     }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

6004     {
6005         \begin { minipage } [ \l_@@_vpos_of_block_tl ]
6006         { \l_@@_col_width_dim }
6007         \str_case:Vn \l_@@_hpos_block_str
6008         {
6009             c \centering
6010             r \raggedleft
6011             l \raggedright
6012         }
6013         #5
6014         \end { minipage }
6015     }
6016     {
6017         \use:x
6018         {
6019             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
6020             { @ { } \l_@@_hpos_block_str @ { } }
6021         }
6022         #5
6023         \end { tabular }
6024     }
6025 }
6026 {
6027     \c_math_toggle_token
6028     \use:x
6029     {
6030         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
6031         { @ { } \l_@@_hpos_block_str @ { } }
6032     }
6033     #5
6034     \end { array }
6035     \c_math_toggle_token
6036 }
6037 \group_end:
6038 }
6039 \bool_if:NT \g_@@_rotate_bool
6040 {
6041     \box_grotate:cn
6042     { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6043     { 90 }
6044     \bool_gset_false:N \g_@@_rotate_bool
6045 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

6046     \int_compare:nNnT { #2 } = 1
6047     {
6048         \dim_gset:Nn \g_@@_blocks_wd_dim
6049         {
6050             \dim_max:nn
6051             \g_@@_blocks_wd_dim
6052             {
6053                 \box_wd:c
6054                 { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6055             }
6056         }
6057     }

```

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.

```

6058 \int_compare:nNnT { #1 } = 1
6059 {
6060   \dim_gset:Nn \g_@@_blocks_ht_dim
6061   {
6062     \dim_max:nn
6063     \g_@@_blocks_ht_dim
6064     {
6065       \box_ht:c
6066       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6067     }
6068   }
6069   \dim_gset:Nn \g_@@_blocks_dp_dim
6070   {
6071     \dim_max:nn
6072     \g_@@_blocks_dp_dim
6073     {
6074       \box_dp:c
6075       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6076     }
6077   }
6078 }
6079 \seq_gput_right:Nx \g_@@_blocks_seq
6080 {
6081   \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.

```

6082 { \exp_not:n { #3 } , \l_@@_hpos_block_str }
6083 {
6084   \box_use_drop:c
6085   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6086 }
6087 }
6088 }

```

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.

```

6089 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
6090 {
6091   \seq_gput_right:Nx \g_@@_blocks_seq
6092   {
6093     \l_tmpa_tl
6094     { \exp_not:n { #3 } }
6095     \exp_not:n
6096     {
6097       {
6098         \bool_if:NTF \l_@@_NiceTabular_bool
6099         {
6100           \group_begin:
6101           \bool_if:NF \l_@@_respect_arraystretch_bool
6102           { \cs_set:Npn \arraystretch { 1 } }
6103           \dim_zero:N \extrarowheight
6104           #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).

```

6105         \bool_if:NT \g_@@_rotate_bool
6106         { \str_set:Nn \l_@@_hpos_block_str c }
6107     \use:x
6108     {
6109         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
6110         { @ { } \l_@@_hpos_block_str @ { } }
6111     }
6112     #5
6113 \end { tabular }
6114 \group_end:
6115 }
6116 {
6117     \group_begin:
6118     \bool_if:NF \l_@@_respect_arraystretch_bool
6119     { \cs_set:Npn \arraystretch { 1 } }
6120     \dim_zero:N \extrarowheight
6121     #4
6122     \bool_if:NT \g_@@_rotate_bool
6123     { \str_set:Nn \l_@@_hpos_block_str c }
6124     \c_math_toggle_token
6125     \use:x
6126     {
6127         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
6128         { @ { } \l_@@_hpos_block_str @ { } }
6129     }
6130     #5
6131     \end { array }
6132     \c_math_toggle_token
6133     \group_end:
6134 }
6135 }
6136 }
6137 }
6138 }

```

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).

```

6139 \keys_define:nn { NiceMatrix / Block / SecondPass }
6140 {
6141     tikz .code:n =
6142         \bool_if:NTF \c_@@_tikz_loaded_bool
6143         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
6144         { \@@_error:n { tikz-key-without~tikz } } ,
6145     tikz .value_required:n = true ,
6146     fill .tl_set:N = \l_@@_fill_tl ,
6147     fill .value_required:n = true ,
6148     draw .tl_set:N = \l_@@_draw_tl ,
6149     draw .default:n = default ,
6150     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6151     rounded-corners .default:n = 4 pt ,
6152     color .code:n =
6153         \@@_color:n { #1 }
6154         \tl_set:Nn \l_@@_draw_tl { #1 } ,
6155     color .value_required:n = true ,
6156     borders .clist_set:N = \l_@@_borders_clist ,
6157     borders .value_required:n = true ,
6158     hvlines .meta:n = { vlines , hlines } ,
6159     vlines .bool_set:N = \l_@@_vlines_block_bool ,
6160     vlines .default:n = true ,

```



```

6161 hlines .bool_set:N = \l_@@_hlines_block_bool,
6162 hlines .default:n = true ,
6163 line-width .dim_set:N = \l_@@_line_width_dim ,
6164 line-width .value_required:n = true ,
6165 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6166 l .value_forbidden:n = true ,
6167 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6168 r .value_forbidden:n = true ,
6169 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6170 c .value_forbidden:n = true ,
6171 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
6172         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6173 L .value_forbidden:n = true ,
6174 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6175         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6176 R .value_forbidden:n = true ,
6177 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6178         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6179 C .value_forbidden:n = true ,
6180 t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
6181 t .value_forbidden:n = true ,
6182 b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
6183 b .value_forbidden:n = true ,
6184 name .tl_set:N = \l_@@_block_name_str ,
6185 name .value_required:n = true ,
6186 name .initial:n = ,
6187 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6188 respect-arraystretch .default:n = true ,
6189 v-center .bool_set:N = \l_@@_v_center_bool ,
6190 v-center .default:n = true ,
6191 v-center .initial:n = false ,
6192 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
6193 }

```

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.

```

6194 \cs_new_protected:Npn \@@_draw_blocks:
6195 {
6196   \cs_set_eq:NN \ialign \@@_old_ialign:
6197   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
6198 }
6199 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
6200 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6201   \int_zero_new:N \l_@@_last_row_int
6202   \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.

```

6203   \int_compare:nNnTF { #3 } > { 99 }
6204   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
6205   { \int_set:Nn \l_@@_last_row_int { #3 } }
6206   \int_compare:nNnTF { #4 } > { 99 }
6207   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
6208   { \int_set:Nn \l_@@_last_col_int { #4 } }

```

```

6209 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6210 {
6211   \int_compare:nTF
6212     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6213     {
6214       \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
6215       \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
6216       \group_begin:
6217       \globaldefs = 1
6218       \@@_msg_redirect_name:nn { columns-not-used } { none }
6219       \group_end:
6220     }
6221     { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6222   }
6223   {
6224     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6225     { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6226     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6227   }
6228 }
6229 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6230 {

```

The group is for the keys.

```

6231   \group_begin:
6232   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }

```

We restrict the use of the key v-center to the case of a mono-row block.

```

6233   \bool_if:NT \l_@@_v_center_bool
6234   {
6235     \int_compare:nNnF { #1 } = { #3 }
6236     {
6237       \@@_error:n { Wrong-use-of-v-center }
6238       \bool_set_false:N \l_@@_v_center_bool
6239     }
6240   }
6241   \bool_if:NT \l_@@_vlines_block_bool
6242   {
6243     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6244     {
6245       \@@_vlines_block:nnn
6246       { \exp_not:n { #5 } }
6247       { #1 - #2 }
6248       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6249     }
6250   }
6251   \bool_if:NT \l_@@_hlines_block_bool
6252   {
6253     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6254     {
6255       \@@_hlines_block:nnn
6256       { \exp_not:n { #5 } }
6257       { #1 - #2 }
6258       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6259     }
6260   }
6261   \bool_if:nT
6262   { ! \l_@@_vlines_block_bool && ! \l_@@_hlines_block_bool }
6263   {

```

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).

```

6264   \seq_gput_left:Nx \g_@@_pos_of_blocks_seq

```

```

6265         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6266     }
6267 \tl_if_empty:NF \l_@@_draw_tl
6268 {
6269     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6270     {
6271         \@@_stroke_block:nnn
6272         { \exp_not:n { #5 } }
6273         { #1 - #2 }
6274         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6275     }
6276     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6277     { { #1 } { #2 } { #3 } { #4 } }
6278 }
6279 \clist_if_empty:NF \l_@@_borders_clist
6280 {
6281     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6282     {
6283         \@@_stroke_borders_block:nnn
6284         { \exp_not:n { #5 } }
6285         { #1 - #2 }
6286         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6287     }
6288 }
6289 \tl_if_empty:NF \l_@@_fill_tl
6290 {
6291     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6292     {
6293         \exp_not:N \roundedrectanglecolor
6294         \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
6295             { \l_@@_fill_tl }
6296             { { \l_@@_fill_tl } }
6297             { #1 - #2 }
6298             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6299             { \dim_use:N \l_@@_rounded_corners_dim }
6300         ]
6301     }
6302 }
6303 \seq_if_empty:NF \l_@@_tikz_seq
6304 {
6305     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6306     {
6307         \@@_block_tikz:nnnnn
6308         { #1 }
6309         { #2 }
6310         { \int_use:N \l_@@_last_row_int }
6311         { \int_use:N \l_@@_last_col_int }
6312         { \seq_use:Nn \l_@@_tikz_seq { , } }
6313     }
6314 }
6315 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6316 {
6317     \tl_gput_right:Nx \g_@@_internal_code_after_tl
6318     {
6319         \@@_actually_diagbox:nnnnnn
6320         { #1 }
6321         { #2 }
6322         { \int_use:N \l_@@_last_row_int }
6323         { \int_use:N \l_@@_last_col_int }
6324         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6325     }
6326 }

```

```

6326 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6327 \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.

```

6328 \pgfpicture
6329 \pgfrememberpicturepositiononpagetrue
6330 \pgf@relevantforpicturesizefalse
6331 \@@_qpoint:n { row - #1 }
6332 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6333 \@@_qpoint:n { col - #2 }
6334 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6335 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6336 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6337 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6338 \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.

```

6339 \@@_pgf_rect_node:nnnnn
6340 { \@@_env: - #1 - #2 - block }
6341 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6342 \str_if_empty:NF \l_@@_block_name_str
6343 {
6344 \pgfnodealias
6345 { \@@_env: - \l_@@_block_name_str }
6346 { \@@_env: - #1 - #2 - block }
6347 \str_if_empty:NF \l_@@_name_str
6348 {
6349 \pgfnodealias
6350 { \l_@@_name_str - \l_@@_block_name_str }
6351 { \@@_env: - #1 - #2 - block }
6352 }
6353 }

```

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.

```

6354 \bool_if:NF \l_@@_hpos_of_block_cap_bool
6355 {
6356 \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.

```

6357         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6358     {

```

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.

```

6359         \cs_if_exist:cT
6360         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6361     {
6362         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6363     {
6364         \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
6365         \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
6366     }
6367     }
6368 }

```

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.

```

6369     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
6370     {
6371         \@@_qpoint:n { col - #2 }
6372         \dim_set_eq:NN \l_tmpb_dim \pgf@x
6373     }
6374     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
6375     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6376     {
6377         \cs_if_exist:cT
6378         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6379     {
6380         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6381     {
6382         \pgfpointanchor
6383         { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6384         { east }
6385         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
6386     }
6387     }
6388 }
6389     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
6390     {
6391         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6392         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6393     }
6394     \@@_pgf_rect_node:nnnnn
6395     { \@@_env: - #1 - #2 - block - short }
6396     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6397 }

```

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.

```

6398     \bool_if:NT \l_@@_medium_nodes_bool
6399     {
6400         \@@_pgf_rect_node:nnn
6401         { \@@_env: - #1 - #2 - block - medium }
6402         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
6403         {
6404             \pgfpointanchor
6405             { \@@_env:
6406               - \int_use:N \l_@@_last_row_int
6407               - \int_use:N \l_@@_last_col_int - medium
6408             }
6409             { south-east }

```

```

6410     }
6411 }

```

Now, we will put the label of the block beginning with the case of a \Block of one row.

```

6412 \bool_if:nTF
6413 { \int_compare:nNn { #1 } = { #3 } && ! \l_@@_v_center_bool }
6414 {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

6415     \int_compare:nNnTF { #1 } = 0
6416     { \l_@@_code_for_first_row_tl }
6417     {
6418         \int_compare:nNnT { #1 } = \l_@@_last_row_int
6419         \l_@@_code_for_last_row_tl
6420     }

```

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.

```

6421     \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in \pgf@x) the *x*-value of the center of the block.

```

6422 \pgfpointanchor
6423 {
6424     \@@_env: - #1 - #2 - block
6425     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6426 }
6427 {
6428     \str_case:Vn \l_@@_hpos_block_str
6429     {
6430         c { center }
6431         l { west }
6432         r { east }
6433     }
6434 }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

6435 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
6436 \pgfset { inner-sep = \c_zero_dim }
6437 \pgfnode
6438 { rectangle }
6439 {
6440     \str_case:Vn \l_@@_hpos_block_str
6441     {
6442         c { base }
6443         l { base-west }
6444         r { base-east }
6445     }
6446 }
6447 { \box_use_drop:N \l_@@_cell_box } { } { }
6448 }

```

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).

```

6449 {

```

If we are in the first column, we must put the block as if it was with the key *r*.

```

6450     \int_compare:nNnT { #2 } = 0
6451     { \str_set:Nn \l_@@_hpos_block_str r }
6452     \bool_if:nT \g_@@_last_col_found_bool
6453     {
6454         \int_compare:nNnT { #2 } = \g_@@_col_total_int
6455         { \str_set:Nn \l_@@_hpos_block_str l }
6456     }
6457 \pgftransformshift

```

```

6458     {
6459         \pgfpointanchor
6460         {
6461             \@@_env: - #1 - #2 - block
6462             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6463         }
6464         {
6465             \str_case:Vn \l_@@_hpos_block_str
6466             {
6467                 c { center }
6468                 l { west }
6469                 r { east }
6470             }
6471         }
6472     }
6473     \pgfset { inner-sep = \c_zero_dim }
6474     \pgfnode
6475     { rectangle }
6476     {
6477         \str_case:Vn \l_@@_hpos_block_str
6478         {
6479             c { center }
6480             l { west }
6481             r { east }
6482         }
6483     }
6484     { \box_use_drop:N \l_@@_cell_box } { } { } { }
6485 }
6486 \endpgfpicture
6487 \group_end:
6488 }

```

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).

```

6489 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
6490 {
6491     \group_begin:
6492     \tl_clear:N \l_@@_draw_tl
6493     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6494     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
6495     \pgfpicture
6496     \pgfrememberpicturepositiononpagetrue
6497     \pgf@relevantforpicturesizefalse
6498     \tl_if_empty:NF \l_@@_draw_tl
6499     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

6500         \str_if_eq:VnTF \l_@@_draw_tl { default }
6501         { \CT@arc@ }
6502         { \@@_color:V \l_@@_draw_tl }
6503     }
6504     \pgfsetcornersarced
6505     {
6506         \pgfpoint
6507         { \dim_use:N \l_@@_rounded_corners_dim }
6508         { \dim_use:N \l_@@_rounded_corners_dim }
6509     }
6510     \@@_cut_on_hyphen:w #2 \q_stop
6511     \bool_lazy_and:nnT
6512     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
6513     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }

```

```

6514 {
6515   \@@_qpoint:n { row - \l_tmpa_tl }
6516   \dim_set:Nn \l_tmpb_dim { \pgf@y }
6517   \@@_qpoint:n { col - \l_tmpb_tl }
6518   \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
6519   \@@_cut_on_hyphen:w #3 \q_stop
6520   \int_compare:nNnT \l_tmpa_tl > \c@iRow
6521     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
6522   \int_compare:nNnT \l_tmpb_tl > \c@jCol
6523     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
6524   \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
6525   \dim_set:Nn \l_tmpa_dim { \pgf@y }
6526   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6527   \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
6528   \pgfpathrectanglecorners
6529     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6530     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6531   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

6532   \pgfusepath { stroke }
6533 }
6534 \endpgfpicture
6535 \group_end:
6536 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

6537 \keys_define:nn { NiceMatrix / BlockStroke }
6538 {
6539   color .tl_set:N = \l_@@_draw_tl ,
6540   draw .tl_set:N = \l_@@_draw_tl ,
6541   draw .default:n = default ,
6542   line-width .dim_set:N = \l_@@_line_width_dim ,
6543   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6544   rounded-corners .default:n = 4 pt
6545 }

```

The first argument of `\@@_vlines_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).

```

6546 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
6547 {
6548   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6549   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6550   \@@_cut_on_hyphen:w #2 \q_stop
6551   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6552   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6553   \@@_cut_on_hyphen:w #3 \q_stop
6554   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6555   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6556   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
6557   {
6558     \use:x
6559     {
6560       \@@_vline:n
6561       {
6562         position = ##1 ,
6563         start = \l_@@_tmpc_tl ,
6564         end = \int_eval:n { \l_tmpa_tl - 1 }
6565       }
6566     }
6567   }
6568 }
6569 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3

```



```

6570 {
6571   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6572   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6573   \@@_cut_on_hyphen:w #2 \q_stop
6574   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6575   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6576   \@@_cut_on_hyphen:w #3 \q_stop
6577   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6578   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6579   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
6580   {
6581     \use:x
6582     {
6583       \@@_hline:n
6584       {
6585         position = ##1 ,
6586         start = \l_@@_tmpd_tl ,
6587         end = \int_eval:n { \l_tmpb_tl - 1 } ,
6588         total-width = \arrayrulewidth
6589       }
6590     }
6591   }
6592 }

```

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).

```

6593 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
6594 {
6595   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6596   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6597   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
6598   { \@@_error:n { borders~forbidden } }
6599   {
6600     \tl_clear_new:N \l_@@_borders_tikz_tl
6601     \keys_set:nV
6602     { NiceMatrix / OnlyForTikzInBorders }
6603     \l_@@_borders_clist
6604     \@@_cut_on_hyphen:w #2 \q_stop
6605     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6606     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6607     \@@_cut_on_hyphen:w #3 \q_stop
6608     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6609     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6610     \@@_stroke_borders_block_i:
6611   }
6612 }
6613 \hook_gput_code:nnn { begindocument } { . }
6614 {
6615   \cs_new_protected:Npx \@@_stroke_borders_block_i:
6616   {
6617     \c_@@_pgfortikzpicture_tl
6618     \@@_stroke_borders_block_ii:
6619     \c_@@_endpgfortikzpicture_tl
6620   }
6621 }
6622 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
6623 {
6624   \pgfrememberpicturepositiononpagetrue
6625   \pgf@relevantforpicturesizefalse
6626   \CT@arc@
6627   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

```

6628 \clist_if_in:NnT \l_@@_borders_clist { right }
6629 { \@@_stroke_vertical:n \l_tmpb_tl }
6630 \clist_if_in:NnT \l_@@_borders_clist { left }
6631 { \@@_stroke_vertical:n \l_@@_tmpd_tl }
6632 \clist_if_in:NnT \l_@@_borders_clist { bottom }
6633 { \@@_stroke_horizontal:n \l_tmpa_tl }
6634 \clist_if_in:NnT \l_@@_borders_clist { top }
6635 { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
6636 }
6637 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
6638 {
6639   tikz .code:n =
6640     \cs_if_exist:NTF \tikzpicture
6641     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
6642     { \@@_error:n { tikz-in~borders~without~tikz } } ,
6643   tikz .value_required:n = true ,
6644   top .code:n = ,
6645   bottom .code:n = ,
6646   left .code:n = ,
6647   right .code:n = ,
6648   unknown .code:n = \@@_error:n { bad~border }
6649 }

```

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).

```

6650 \cs_new_protected:Npn \@@_stroke_vertical:n #1
6651 {
6652   \@@_qpoint:n \l_@@_tmpc_tl
6653   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6654   \@@_qpoint:n \l_tmpa_tl
6655   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6656   \@@_qpoint:n { #1 }
6657   \tl_if_empty:NTF \l_@@_borders_tikz_tl
6658   {
6659     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
6660     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
6661     \pgfusepathqstroke
6662   }
6663   {
6664     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
6665     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
6666   }
6667 }

```

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).

```

6668 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
6669 {
6670   \@@_qpoint:n \l_@@_tmpd_tl
6671   \clist_if_in:NnTF \l_@@_borders_clist { left }
6672   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
6673   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
6674   \@@_qpoint:n \l_tmpb_tl
6675   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
6676   \@@_qpoint:n { #1 }
6677   \tl_if_empty:NTF \l_@@_borders_tikz_tl
6678   {
6679     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
6680     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6681     \pgfusepathqstroke
6682   }
6683   {
6684     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }

```

```

6685         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
6686     }
6687 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

6688 \keys_define:nn { NiceMatrix / BlockBorders }
6689 {
6690     borders .clist_set:N = \l_@@_borders_clist ,
6691     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6692     rounded-corners .default:n = 4 pt ,
6693     line-width .dim_set:N = \l_@@_line_width_dim ,
6694 }

```

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.

```

6695 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
6696 {
6697     \begin { tikzpicture }
6698     \clist_map_inline:nn { #5 }
6699     {
6700         \path [ ##1 ]
6701             ( #1 -| #2 )
6702             rectangle
6703             ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
6704     }
6705     \end { tikzpicture }
6706 }

```

How to draw the dotted lines transparently

```

6707 \cs_set_protected:Npn \@@_renew_matrix:
6708 {
6709     \RenewDocumentEnvironment { pmatrix } { } {
6710         { \pNiceMatrix }
6711         { \endpNiceMatrix }
6712     }
6713     \RenewDocumentEnvironment { vmatrix } { } {
6714         { \vNiceMatrix }
6715         { \endvNiceMatrix }
6716     }
6717     \RenewDocumentEnvironment { Vmatrix } { } {
6718         { \VNiceMatrix }
6719         { \endVNiceMatrix }
6720     }
6721     \RenewDocumentEnvironment { bmatrix } { } {
6722         { \bNiceMatrix }
6723         { \endbNiceMatrix }
6724     }
6725     \RenewDocumentEnvironment { Bmatrix } { } {
6726         { \BNiceMatrix }
6727         { \endBNiceMatrix }
6728     }
6729 }

```

Automatic arrays

```

6725 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
6726 {
6727     \int_set:Nn \l_@@_nb_rows_int { #1 }
6728     \int_set:Nn \l_@@_nb_cols_int { #2 }
6729 }

```

We will extract the potential keys `columns-type`, `l`, `c`, `r` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

6730 \keys_define:nn { NiceMatrix / Auto }
6731 {
6732   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
6733   columns-type .value_required:n = true ,
6734   l .meta:n = { columns-type = l } ,
6735   r .meta:n = { columns-type = r } ,
6736   c .meta:n = { columns-type = c }
6737 }
6738 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
6739 {
6740   \int_zero_new:N \l_@@_nb_rows_int
6741   \int_zero_new:N \l_@@_nb_cols_int
6742   \@@_set_size:n #4 \q_stop

```

The group is for the protection of the keys.

```

6743   \group_begin:
6744   \bool_set_true:N \l_@@_Matrix_bool
6745   \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl

```

We nullify the command \@@_transform_preamble: because we will provide a preamble which is yet transformed (by using \l_@@_columns_type_tl which is yet nicematrix-ready).

```

6746   \cs_set_eq:NN \@@_transform_preamble: \prg_do_nothing:
6747   \use:x
6748   {
6749     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
6750     {
6751       * { \int_use:N \l_@@_nb_cols_int }
6752       { \exp_not:N \l_@@_columns_type_tl }
6753     }
6754     [ \exp_not:N \l_tmpa_tl ]
6755   }
6756   \int_compare:nNnT \l_@@_first_row_int = 0
6757   {
6758     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6759     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6760     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6761   }
6762   \prg_replicate:nn \l_@@_nb_rows_int
6763   {
6764     \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).

```

6765     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
6766     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6767   }
6768   \int_compare:nNnT \l_@@_last_row_int > { -2 }
6769   {
6770     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6771     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6772     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6773   }
6774   \end { NiceArrayWithDelims }
6775   \group_end:
6776 }
6777 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6778 {
6779   \cs_set_protected:cpn { #1 AutoNiceMatrix }
6780   {
6781     \bool_gset_false:N \g_@@_NiceArray_bool
6782     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6783     \AutoNiceMatrixWithDelims { #2 } { #3 }
6784   }

```

```

6785 }
6786 \@@_define_com:nnn p ( )
6787 \@@_define_com:nnn b [ ]
6788 \@@_define_com:nnn v | |
6789 \@@_define_com:nnn V \ | \ |
6790 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

6791 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
6792 {
6793   \group_begin:
6794   \bool_gset_true:N \g_@@_NiceArray_bool
6795   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6796   \group_end:
6797 }

```

The redefinition of the command `\dotfill`

```

6798 \cs_set_eq:NN \@@_old_dotfill \dotfill
6799 \cs_new_protected:Npn \@@_dotfill:
6800 {

```

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}`).

```

6801   \@@_old_dotfill
6802   \bool_if:NT \l_@@_NiceTabular_bool
6803     { \group_insert_after:N \@@_dotfill_ii: }
6804     { \group_insert_after:N \@@_dotfill_i: }
6805 }
6806 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
6807 \cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

6808 \cs_new_protected:Npn \@@_dotfill_iii:
6809 { \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.

```

6810 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6811 {
6812   \tl_gput_right:Nx \g_@@_internal_code_after_tl
6813   {
6814     \@@_actually_diagbox:nnnnnn
6815     { \int_use:N \c@iRow }
6816     { \int_use:N \c@jCol }
6817     { \int_use:N \c@iRow }
6818     { \int_use:N \c@jCol }
6819     { \exp_not:n { #1 } }
6820     { \exp_not:n { #2 } }
6821   }

```

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`.

```

6822   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6823   {
6824     { \int_use:N \c@iRow }
6825     { \int_use:N \c@jCol }
6826     { \int_use:N \c@iRow }
6827     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```
6828     { }
6829   }
6830 }
```

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.

```
6831 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
6832 {
6833   \pgfpicture
6834   \pgf@relevantforpicturesizefalse
6835   \pgfrememberpicturepositiononpagetrue
6836   \@@_qpoint:n { row - #1 }
6837   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6838   \@@_qpoint:n { col - #2 }
6839   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6840   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6841   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
6842   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6843   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
6844   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6845   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6846   {
```

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.

```
6847     \CT@arc@
6848     \pgfsetroundcap
6849     \pgfusepathqstroke
6850   }
6851   \pgfset { inner~sep = 1 pt }
6852   \pgfscope
6853   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6854   \pgfnode { rectangle } { south-west }
6855   {
6856     \begin { minipage } { 20 cm }
6857     \@@_math_toggle_token: #5 \@@_math_toggle_token:
6858     \end { minipage }
6859   }
6860   { }
6861   { }
6862   \endpgfscope
6863   \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6864   \pgfnode { rectangle } { north-east }
6865   {
6866     \begin { minipage } { 20 cm }
6867     \raggedleft
6868     \@@_math_toggle_token: #6 \@@_math_toggle_token:
6869     \end { minipage }
6870   }
6871   { }
6872   { }
6873   \endpgfpicture
6874 }
```

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.

```

6875 \keys_define:nn { NiceMatrix }
6876 {
6877   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6878   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6879 }
6880 \keys_define:nn { NiceMatrix / CodeAfter }
6881 {
6882   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6883   sub-matrix .value_required:n = true ,
6884   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6885   delimiters / color .value_required:n = true ,
6886   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6887   rules .value_required:n = true ,
6888   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
6889 }

```

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. 125.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

6890 \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 `\`.

```

6891 \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`.

```

6892 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
6893 {
6894   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6895   \@@_CodeAfter_iv:n
6896 }

```

We catch the argument of the command `\end` (in `#1`).

```

6897 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
6898 {

```

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.

```

6899   \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`.

```

6900   {
6901     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6902     \@@_CodeAfter_ii:n
6903   }
6904 }

```

The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_internal_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter ($($, $[$, $\{$, $)$, $]$ or $\}$). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```

6905 \cs_new_protected:Npn \l_@@_delimiter:nnn #1 #2 #3
6906 {
6907   \pgfpicture
6908   \pgfrememberpicturepositiononpagetrue
6909   \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.

```

6910   \l_@@_qpoint:n { row - 1 }
6911   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6912   \l_@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
6913   \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).

```

6914   \bool_if:nTF { #3 }
6915   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6916   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6917   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6918   {
6919     \cs_if_exist:cT
6920     { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
6921     {
6922       \pgfpointanchor
6923       { \l_@@_env: - ##1 - #2 }
6924       { \bool_if:nTF { #3 } { west } { east } }
6925       \dim_set:Nn \l_tmpa_dim
6926       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6927     }
6928   }

```

Now we can put the delimiter with a node of PGF.

```

6929   \pgfset { inner~sep = \c_zero_dim }
6930   \dim_zero:N \nulldelimiterspace
6931   \pgftransformshift
6932   {
6933     \pgfpoint
6934     { \l_tmpa_dim }
6935     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6936   }
6937   \pgfnode
6938   { rectangle }
6939   { \bool_if:nTF { #3 } { east } { west } }
6940   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

6941     \nullfont
6942     \c_math_toggle_token
6943     \l_@@_color:V \l_@@_delimiters_color_tl
6944     \bool_if:nTF { #3 } { \left #1 } { \left . }
6945     \vcenter
6946     {
6947       \nullfont
6948       \hrule \height
6949       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6950       \depth \c_zero_dim
6951       \width \c_zero_dim
6952     }
6953     \bool_if:nTF { #3 } { \right . } { \right #1 }
6954     \c_math_toggle_token

```



```

6955     }
6956     { }
6957     { }
6958 \endpgfpicture
6959 }

```

The command \SubMatrix

```

6960 \keys_define:nn { NiceMatrix / sub-matrix }
6961 {
6962   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6963   extra-height .value_required:n = true ,
6964   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6965   left-xshift .value_required:n = true ,
6966   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6967   right-xshift .value_required:n = true ,
6968   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6969   xshift .value_required:n = true ,
6970   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6971   delimiters / color .value_required:n = true ,
6972   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6973   slim .default:n = true ,
6974   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6975   hlines .default:n = all ,
6976   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6977   vlines .default:n = all ,
6978   hvlines .meta:n = { hlines, vlines } ,
6979   hvlines .value_forbidden:n = true ,
6980 }
6981 \keys_define:nn { NiceMatrix }
6982 {
6983   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6984   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6985   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6986   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6987   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6988   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6989 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

6990 \keys_define:nn { NiceMatrix / SubMatrix }
6991 {
6992   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6993   delimiters / color .value_required:n = true ,
6994   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6995   hlines .default:n = all ,
6996   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6997   vlines .default:n = all ,
6998   hvlines .meta:n = { hlines, vlines } ,
6999   hvlines .value_forbidden:n = true ,
7000   name .code:n =
7001     \tl_if_empty:nTF { #1 }
7002     { \@@_error:n { Invalid-name } }
7003     {
7004       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7005       {
7006         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7007         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7008         {
7009           \str_set:Nn \l_@@_submatrix_name_str { #1 }
7010           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7011         }
7012       }
7013     }

```

```

7012     }
7013     { \@@_error:n { Invalid-name } }
7014   } ,
7015   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7016   rules .value_required:n = true ,
7017   code .tl_set:N = \l_@@_code_tl ,
7018   code .value_required:n = true ,
7019   name .value_required:n = true ,
7020   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
7021 }

7022 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
7023 {
7024   \peek_remove_spaces:n
7025   {
7026     \tl_gput_right:Nn \g_@@_internal_code_after_tl
7027     { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
7028     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
7029   }
7030 }

7031 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
7032 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7033 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

7034 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
7035 {
7036   \seq_gput_right:Nx \g_@@_submatrix_seq
7037   {
We use \str_if_eq:nnTF because it is fully expandable.
7038     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
7039     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
7040     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
7041     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
7042   }
7043 }

```

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.

```

7044 \hook_gput_code:nnn { begindocument } { . }
7045 {
7046   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
7047   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
7048   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
7049   {
7050     \peek_remove_spaces:n
7051     {
7052       \@@_sub_matrix:nnnnnnn

```

```

7053         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
7054     }
7055 }
7056 }

```

The following macro will compute `\l_@@_first_i_tl`, `\l_@@_first_j_tl`, `\l_@@_last_i_tl` and `\l_@@_last_j_tl` from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

7057 \NewDocumentCommand \@@_compute_i_j:nn
7058 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
7059 { \@@_compute_i_j:nnnn #1 #2 }
7060 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
7061 {
7062     \tl_set:Nn \l_@@_first_i_tl { #1 }
7063     \tl_set:Nn \l_@@_first_j_tl { #2 }
7064     \tl_set:Nn \l_@@_last_i_tl { #3 }
7065     \tl_set:Nn \l_@@_last_j_tl { #4 }
7066     \tl_if_eq:NnT \l_@@_first_i_tl { last }
7067     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
7068     \tl_if_eq:NnT \l_@@_first_j_tl { last }
7069     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
7070     \tl_if_eq:NnT \l_@@_last_i_tl { last }
7071     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
7072     \tl_if_eq:NnT \l_@@_last_j_tl { last }
7073     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
7074 }
7075 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
7076 {
7077     \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

7078 \@@_compute_i_j:nn { #2 } { #3 }
7079 \bool_lazy_or:nnTF
7080 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7081 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7082 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
7083 {
7084     \str_clear_new:N \l_@@_submatrix_name_str
7085     \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
7086     \pgfpicture
7087     \pgfrememberpicturepositiononpagetrue
7088     \pgf@relevantforpicturesizefalse
7089     \pgfset { inner-sep = \c_zero_dim }
7090     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7091     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currying.

```

7092 \bool_if:NTF \l_@@_submatrix_slim_bool
7093 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
7094 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
7095 {
7096     \cs_if_exist:cT
7097     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7098     {
7099         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7100         \dim_set:Nn \l_@@_x_initial_dim
7101         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7102     }
7103     \cs_if_exist:cT
7104     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7105     {
7106         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7107         \dim_set:Nn \l_@@_x_final_dim

```

```

7108         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7109     }
7110 }
7111 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
7112 { \@@_error:nn { Impossible-delimiter } { left } }
7113 {
7114     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
7115     { \@@_error:nn { Impossible-delimiter } { right } }
7116     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
7117 }
7118 \endpgfpicture
7119 }
7120 \group_end:
7121 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

7122 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
7123 {
7124     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
7125     \dim_set:Nn \l_@@_y_initial_dim
7126     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
7127     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
7128     \dim_set:Nn \l_@@_y_final_dim
7129     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
7130     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
7131     {
7132         \cs_if_exist:cT
7133         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
7134         {
7135             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
7136             \dim_set:Nn \l_@@_y_initial_dim
7137             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
7138         }
7139         \cs_if_exist:cT
7140         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
7141         {
7142             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
7143             \dim_set:Nn \l_@@_y_final_dim
7144             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
7145         }
7146     }
7147     \dim_set:Nn \l_tmpa_dim
7148     {
7149         \l_@@_y_initial_dim - \l_@@_y_final_dim +
7150         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
7151     }
7152     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

7153 \group_begin:
7154 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7155 \@@_set_CT@arc@:V \l_@@_rules_color_tl
7156 \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`.

```

7157 \seq_map_inline:Nn \g_@@_cols_vlism_seq
7158 {
7159     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
7160     {
7161         \int_compare:nNnT
7162         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```

7163         {
First, we extract the value of the abscissa of the rule we have to draw.
7164         \@@_qpoint:n { col - ##1 }
7165         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7166         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7167         \pgfusepathqstroke
7168     }
7169 }
7170 }

```

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.

```

7171 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
7172 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
7173 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
7174 {
7175     \bool_lazy_and:nnTF
7176     { \int_compare_p:nNn { ##1 } > 0 }
7177     {
7178         \int_compare_p:nNn
7179         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7180     {
7181         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7182         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7183         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7184         \pgfusepathqstroke
7185     }
7186     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
7187 }

```

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.

```

7188 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7189 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7190 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
7191 {
7192     \bool_lazy_and:nnTF
7193     { \int_compare_p:nNn { ##1 } > 0 }
7194     {
7195         \int_compare_p:nNn
7196         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7197     {
7198         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7199     \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
7200     \dim_set:Nn \l_tmpa_dim
7201     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7202     \str_case:nn { #1 }
7203     {
7204         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7205         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7206         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7207     }
7208     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

7209     \dim_set:Nn \l_tmpb_dim
7210     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7211     \str_case:nn { #2 }
7212     {
7213         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }

```

```

7214         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7215         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7216     }
7217     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7218     \pgfusepathqstroke
7219     \group_end:
7220 }
7221 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7222 }

```

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).

```

7223 \str_if_empty:NF \l_@@_submatrix_name_str
7224 {
7225     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
7226     \l_@@_x_initial_dim \l_@@_y_initial_dim
7227     \l_@@_x_final_dim \l_@@_y_final_dim
7228 }
7229 \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`.

```

7230 \begin { pgfscope }
7231 \pgftransformshift
7232 {
7233     \pgfpoint
7234     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7235     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7236 }
7237 \str_if_empty:NTF \l_@@_submatrix_name_str
7238 { \@@_node_left:nn #1 { } }
7239 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7240 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7241 \pgftransformshift
7242 {
7243     \pgfpoint
7244     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7245     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7246 }
7247 \str_if_empty:NTF \l_@@_submatrix_name_str
7248 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7249 {
7250     \@@_node_right:nnnn #2
7251     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7252 }
7253 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7254 \flag_clear_new:n { nicematrix }
7255 \l_@@_code_tl
7256 }

```

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`.

```

7257 \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.

```

7258 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
7259 {
7260   \use:e
7261   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpointanchor_i:nn #1 } }
7262 }
```

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 -.

```

7263 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
7264 { #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.

```

7265 \tl_const:Nn \c_@@_integers_alist_tl
7266 {
7267   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7268   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7269   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7270   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7271 }
```

```

7272 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
7273 {
```

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 or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

7274   \tl_if_empty:nTF { #2 }
7275   {
7276     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7277     {
7278       \flag_raise:n { nicematrix }
7279       \int_if_even:nTF { \flag_height:n { nicematrix } }
7280       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7281       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7282     }
7283     { #1 }
7284   }
```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

7285   { \@@_pgfpointanchor_iii:w { #1 } #2 }
7286 }
```

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`).

```

7287 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
7288 {
7289   \str_case:nnF { #1 }
7290   {
7291     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7292     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7293   }
```

Now the case of a node of the form $i-j$.

```

7294     {
7295         \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7296         - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7297     }
7298 }
```

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`).

```

7299 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7300 {
7301     \pgfnode
7302     { rectangle }
7303     { east }
7304     {
7305         \nullfont
7306         \c_math_toggle_token
7307         \@@_color:V \l_@@_delimiters_color_tl
7308         \left #1
7309         \vcenter
7310         {
7311             \nullfont
7312             \hrule \@height \l_tmpa_dim
7313                 \@depth \c_zero_dim
7314                 \@width \c_zero_dim
7315         }
7316         \right .
7317         \c_math_toggle_token
7318     }
7319     { #2 }
7320     { }
7321 }
```

The command `\@@_node_right:nnn` 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.

```

7322 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
7323 {
7324     \pgfnode
7325     { rectangle }
7326     { west }
7327     {
7328         \nullfont
7329         \c_math_toggle_token
7330         \@@_color:V \l_@@_delimiters_color_tl
7331         \left .
7332         \vcenter
7333         {
7334             \nullfont
7335             \hrule \@height \l_tmpa_dim
7336                 \@depth \c_zero_dim
7337                 \@width \c_zero_dim
7338         }
7339         \right #1
7340         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
7341         ^ { \smash { #4 } }
7342         \c_math_toggle_token
7343     }
7344     { #2 }
7345     { }
7346 }
```


Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

7347 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
7348 {
7349   \peek_remove_spaces:n
7350   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
7351 }

7352 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
7353 {
7354   \peek_remove_spaces:n
7355   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
7356 }

7357 \keys_define:nn { NiceMatrix / Brace }
7358 {
7359   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
7360   left-shorten .default:n = true ,
7361   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
7362   shorten .meta:n = { left-shorten , right-shorten } ,
7363   right-shorten .default:n = true ,
7364   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
7365   yshift .value_required:n = true ,
7366   yshift .initial:n = \c_zero_dim ,
7367   color .tl_set:N = \l_tmpa_tl ,
7368   color .value_required:n = true ,
7369   unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
7370 }

```

`#1` is the first cell of the rectangle (with the syntax `i-lj`; `#2` is the last cell of the rectangle; `#3` is the label of the text; `#4` is the optional argument (a list of *key-value* pairs); `#5` is equal to `under` or `over`.

```

7371 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
7372 {
7373   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

7374 \@@_compute_i_j:nn { #1 } { #2 }
7375 \bool_lazy_or:nnTF
7376 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7377 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7378 {
7379   \str_if_eq:nnTF { #5 } { under }
7380   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
7381   { \@@_error:nn { Construct-too-large } { \OverBrace } }
7382 }
7383 {
7384   \tl_clear:N \l_tmpa_tl % added the 2022-02-25
7385   \keys_set:nn { NiceMatrix / Brace } { #4 }
7386   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } } % added the 2022-02-25
7387   \pgfpicture
7388   \pgfrememberpicturerepositiononpagetrue
7389   \pgf@relevantforpicturesizefalse
7390   \bool_if:NT \l_@@_brace_left_shorten_bool
7391   {
7392     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7393     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7394     {
7395       \cs_if_exist:cT
7396       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7397       {
7398         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }

```

```

7399         \dim_set:Nn \l_@@_x_initial_dim
7400         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7401     }
7402 }
7403 }
7404 \bool_lazy_or:nnT
7405 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
7406 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
7407 {
7408     \@@_qpoint:n { col - \l_@@_first_j_tl }
7409     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7410 }
7411 \bool_if:NT \l_@@_brace_right_shorten_bool
7412 {
7413     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
7414     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7415     {
7416         \cs_if_exist:cT
7417         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7418         {
7419             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7420             \dim_set:Nn \l_@@_x_final_dim
7421             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7422         }
7423     }
7424 }
7425 \bool_lazy_or:nnT
7426 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
7427 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
7428 {
7429     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
7430     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7431 }
7432 \pgfset { inner-sep = \c_zero_dim }
7433 \str_if_eq:nnTF { #5 } { under }
7434 { \@@_underbrace_i:n { #3 } }
7435 { \@@_overbrace_i:n { #3 } }
7436 \endpgfpicture
7437 }
7438 \group_end:
7439 }

```

The argument is the text to put above the brace.

```

7440 \cs_new_protected:Npn \@@_overbrace_i:n #1
7441 {
7442     \@@_qpoint:n { row - \l_@@_first_i_tl }
7443     \pgftransformshift
7444     {
7445         \pgfpoint
7446         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7447         { \pgf@y + \l_@@_brace_yshift_dim }
7448     }
7449     \pgfnode
7450     { rectangle }
7451     { south }
7452     {
7453         \vbox_top:n
7454         {
7455             \group_begin:
7456             \everycr { }
7457             \halign
7458             {
7459                 \hfil ## \hfil \crrc
7460                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr

```

```

7461         \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7462         \hbox_to_wd:nn
7463         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7464         { \downbracefill } \cr
7465     }
7466     \group_end:
7467 }
7468 }
7469 { }
7470 { }
7471 }

```

The argument is the text to put under the brace.

```

7472 \cs_new_protected:Npn \@@_underbrace_i:n #1
7473 {
7474     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
7475     \pgftransformshift
7476     {
7477         \pgfpoint
7478         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7479         { \pgf@y - \l_@@_brace_yshift_dim }
7480     }
7481     \pgfnode
7482     { rectangle }
7483     { north }
7484     {
7485         \group_begin:
7486         \everycr { }
7487         \vbox:n
7488         {
7489             \halign
7490             {
7491                 \hfil ## \hfil \crrc
7492                 \hbox_to_wd:nn
7493                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7494                 { \upbracefill } \cr
7495                 \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7496                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7497             }
7498         }
7499         \group_end:
7500     }
7501     { }
7502     { }
7503 }

```

The command \ShowCellNames

```

7504 \NewDocumentCommand \@@_ShowCellNames { }
7505 {
7506     \dim_zero_new:N \g_@@_tmpc_dim
7507     \dim_zero_new:N \g_@@_tmpd_dim
7508     \dim_zero_new:N \g_@@_tmpe_dim
7509     \int_step_inline:nn \c@iRow
7510     {
7511         \begin { pgfpicture }
7512         \@@_qpoint:n { row - ##1 }
7513         \dim_set_eq:NN \l_tmpa_dim \pgf@y
7514         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
7515         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
7516         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }

```

```

7517 \end { pgfpicture }
7518 \int_step_inline:nn \c@jCol
7519 {
7520   \hbox_set:Nn \l_tmpa_box
7521     { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
7522   \begin { pgfpicture }
7523     \@@_qpoint:n { col - ####1 }
7524     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
7525     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
7526     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
7527     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
7528   \end { pgfpicture }
7529   \fp_set:Nn \l_tmpa_fp
7530   {
7531     \fp_min:nn
7532     {
7533       \fp_min:nn
7534         { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
7535         { \dim_ratio:nn { \g_@@_tmpe_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
7536       }
7537     { 1.0 }
7538   }
7539   \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
7540   \pgfpicture
7541   \pgfrememberpicturepositiononpagetrue
7542   \pgf@relevantforpicturesizefalse
7543   \pgftransformshift
7544   {
7545     \pgfpoint
7546       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
7547       { \dim_use:N \g_@@_tmpa_dim }
7548   }
7549   \pgfnode
7550   { rectangle }
7551   { center }
7552   { \box_use:N \l_tmpa_box }
7553   { }
7554   { }
7555   \endpgfpicture
7556 }
7557 }
7558 }

```

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.

```
7559 \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.

```
7560 \bool_new:N \c_@@_footnote_bool
```

```

7561 \msg_new:nnnn { nicematrix } { Unknown-key-for-package }
7562 {
7563   The~key~'\l_keys_key_str'~is-unknown. \\
7564   That~key~will~be-ignored. \\
7565   For~a~list~of~the~available~keys,~type-H~<return>.

```

```

7566 }
7567 {
7568   The~available~keys~are~(in~alphabetic~order):~
7569   footnote,~
7570   footnotehyper,~
7571   messages-for-Overleaf,~
7572   renew-dots,~and
7573   renew-matrix.
7574 }
7575 \keys_define:nn { NiceMatrix / Package }
7576 {
7577   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
7578   renew-dots .value_forbidden:n = true ,
7579   renew-matrix .code:n = \@@_renew_matrix: ,
7580   renew-matrix .value_forbidden:n = true ,
7581   messages-for-Overleaf .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
7582   footnote .bool_set:N = \c_@@_footnote_bool ,
7583   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
7584   unknown .code:n = \@@_error:n { Unknown-key-for~package }
7585 }
7586 \ProcessKeysOptions { NiceMatrix / Package }

7587 \@@_msg_new:nn { footnote~with~footnotehyper~package }
7588 {
7589   You~can't~use~the~option~'footnote'~because~the~package~
7590   footnotehyper~has~already~been~loaded.~
7591   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
7592   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7593   of~the~package~footnotehyper.\\
7594   The~package~footnote~won't~be~loaded.
7595 }
7596 \@@_msg_new:nn { footnotehyper~with~footnote~package }
7597 {
7598   You~can't~use~the~option~'footnotehyper'~because~the~package~
7599   footnote~has~already~been~loaded.~
7600   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
7601   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7602   of~the~package~footnote.\\
7603   The~package~footnotehyper~won't~be~loaded.
7604 }

```

```

7605 \bool_if:NT \c_@@_footnote_bool
7606 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7607   \@ifclassloaded { beamer }
7608   { \bool_set_false:N \c_@@_footnote_bool }
7609   {
7610     \@ifpackageloaded { footnotehyper }
7611     { \@@_error:n { footnote~with~footnotehyper~package } }
7612     { \usepackage { footnote } }
7613   }
7614 }
7615 \bool_if:NT \c_@@_footnotehyper_bool
7616 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7617   \@ifclassloaded { beamer }
7618   { \bool_set_false:N \c_@@_footnote_bool }

```

```

7619 {
7620   \ifpackage{footnote}
7621   { \@@_error:n { footnotehyper~with~footnote-package } }
7622   { \usepackage { footnotehyper } }
7623 }
7624 \bool_set_true:N \c_@@_footnote_bool
7625 }

```

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

```

7626 \bool_if:NTF \c_@@_messages_for_Overleaf_bool
7627 { \str_const:Nn \c_@@_available_keys_str { } }
7628 {
7629   \str_const:Nn \c_@@_available_keys_str
7630   { For~a~list~of~the~available~keys,~type~H~<return>. }
7631 }
7632 \seq_new:N \g_@@_types_of_matrix_seq
7633 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
7634 {
7635   NiceMatrix ,
7636   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
7637 }
7638 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
7639 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries 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`.

```

7640 \cs_new_protected:Npn \@@_error_too_much_cols:
7641 {
7642   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
7643   {
7644     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
7645     { \@@_fatal:n { too-much-cols-for-matrix } }
7646     {
7647       \bool_if:NF \l_@@_last_col_without_value_bool
7648       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
7649     }
7650   }
7651   { \@@_fatal:n { too-much-cols-for-array } }
7652 }

```

The following command must *not* be protected since it's used in an error message.

```

7653 \cs_new:Npn \@@_message_hdotsfor:
7654 {
7655   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
7656   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
7657 }
7658 \@@_msg_new:nn { negative-weight }
7659 {
7660   Negative~weight.\
7661   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
7662   the~value~'#1'~.\
7663   The~absolute~value~will~be~used.
7664 }
7665 \@@_msg_new:nn { last~col~not~used }
7666 {

```

```

7667 Column-not-used.\
7668 The-key~'last-col'~is-in-force-but-you-have-not-used-that-last-column~
7669 in-your~\@@_full_name_env:.~However,~you-can-go-on.
7670 }

7671 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
7672 {
7673   Too-much-columns.\
7674   In-the-row~\int_eval:n { \c@jCol - 1 },~
7675   you-try-to-use-more-columns~
7676   than-allowed-by-your~\@@_full_name_env:.~\@@_message_hdotsfor:\
7677   The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
7678   (plus-the-exterior-columns).~This-error-is-fatal.
7679 }

7680 \@@_msg_new:nn { too-much-cols-for-matrix }
7681 {
7682   Too-much-columns.\
7683   In-the-row~\int_eval:n { \c@jCol - 1 },~
7684   you-try-to-use-more-columns~than-allowed-by-your~
7685   \@@_full_name_env:.~\@@_message_hdotsfor:\ Recall-that-the-maximal~
7686   number-of-columns-for-a-matrix-is-fixed-by-the-LaTeX-counter~
7687   'MaxMatrixCols'.~Its-current-value-is~\int_use:N \c@MaxMatrixCols.~
7688   This-error-is-fatal.
7689 }

```

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`.

```

7690 \@@_msg_new:nn { too-much-cols-for-array }
7691 {
7692   Too-much-columns.\
7693   In-the-row~\int_eval:n { \c@jCol - 1 },~
7694   ~you-try-to-use-more-columns~than-allowed-by-your~
7695   \@@_full_name_env:.~\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
7696   \int_use:N \g_@@_static_num_of_col_int\
7697   ~(plus-the-potential-exterior-ones).~
7698   This-error-is-fatal.
7699 }

7700 \@@_msg_new:nn { hvlines-except-corners }
7701 {
7702   Obsolete-key.\
7703   The-key~'hvlines-except-corners'~is-now-obsolete.~You-should~instead~use~the~
7704   keys~'hvlines'~and~'corners'.\
7705   This-error-is-fatal.
7706 }

7707 \@@_msg_new:nn { columns-not-used }
7708 {
7709   Columns-not-used.\
7710   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
7711   \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\
7712   The-columns-you-did-not-used-won't-be-created.\
7713   We-won't-have-similar-error-till-the-end-of-the-document.
7714 }

7715 \@@_msg_new:nn { in-first-col }
7716 {
7717   Erroneous-use.\
7718   You-can't-use-the-command~#1 in-the-first-column-(number~0)-of-the-array.\
7719   That-command-will-be-ignored.
7720 }

7721 \@@_msg_new:nn { in-last-col }
7722 {
7723   Erroneous-use.\
7724   You-can't-use-the-command~#1 in-the-last-column-(exterior)-of-the-array.\

```

```

7725     That~command~will~be~ignored.
7726 }
7727 \@@_msg_new:nn { in~first~row }
7728 {
7729     Erroneous~use.\\
7730     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
7731     That~command~will~be~ignored.
7732 }
7733 \@@_msg_new:nn { in~last~row }
7734 {
7735     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
7736     That~command~will~be~ignored.
7737 }
7738 \@@_msg_new:nn { double~closing~delimiter }
7739 {
7740     Double~delimiter.\\
7741     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
7742     delimiter.~This~delimiter~will~be~ignored.
7743 }
7744 \@@_msg_new:nn { delimiter~after~opening }
7745 {
7746     Double~delimiter.\\
7747     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
7748     delimiter.~That~delimiter~will~be~ignored.
7749 }
7750 \@@_msg_new:nn { bad~option~for~line~style }
7751 {
7752     Bad~line~style.\\
7753     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
7754     is~'standard'.~That~key~will~be~ignored.
7755 }
7756 \@@_msg_new:nn { Unknown~key~for~rules }
7757 {
7758     Unknown~key.\\
7759     There~is~only~two~keys~available~here:~width~and~color.\\
7760     You~key~'\l_keys_key_str'~will~be~ignored.
7761 }
7762 \@@_msg_new:nnn { Unknown~key~for~custom~line }
7763 {
7764     Unknown~key.\\
7765     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
7766     It~will~be~ignored.  \\
7767     \c_@@_available_keys_str
7768 }
7769 {
7770     The~available~keys~are~(in~alphabetic~order):~
7771     color,~
7772     command,~
7773     dotted,~
7774     letter,~
7775     multiplicity,~
7776     sep~color,~
7777     tikz,~and~total~width.
7778 }
7779 \@@_msg_new:nn { Unknown~key~for~xdots }
7780 {
7781     Unknown~key.\\
7782     As~for~now,~there~is~only~five~keys~available~here:~'color',~'inter',~
7783     'line~style',~'radius',~
7784     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~

```



```

7785     That~key~will~be~ignored.
7786 }
7787 \@@_msg_new:nn { Unknown~key~for~rowcolors }
7788 {
7789     Unknown~key.\\
7790     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
7791     (and~you~try~to~use~'\l_keys_key_str')\\
7792     That~key~will~be~ignored.
7793 }
7794 \@@_msg_new:nn { Construct~too~large }
7795 {
7796     Construct~too~large.\\
7797     Your~command~\token_to_str:N #1
7798     can't~be~drawn~because~your~matrix~is~too~small.\\
7799     That~command~will~be~ignored.
7800 }
7801 \@@_msg_new:nn { ampersand~in~light-syntax }
7802 {
7803     Ampersand~forbidden.\\
7804     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
7805     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
7806 }
7807 \@@_msg_new:nn { double-backslash~in~light-syntax }
7808 {
7809     Double~backslash~forbidden.\\
7810     You~can't~use~\token_to_str:N
7811     \\~to~separate~rows~because~the~key~'light-syntax'~
7812     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
7813     (set~by~the~key~'end-of-row').~This~error~is~fatal.
7814 }
7815 \@@_msg_new:nn { bad~value~for~baseline }
7816 {
7817     Bad~value~for~baseline.\\
7818     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
7819     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
7820     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
7821     the~form~'line-i'.\\
7822     A~value~of~1~will~be~used.
7823 }
7824 \@@_msg_new:nn { Invalid~name }
7825 {
7826     Invalid~name.\\
7827     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
7828     \SubMatrix\ of~your~\@@_full_name_env:.\\
7829     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
7830     This~key~will~be~ignored.
7831 }
7832 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
7833 {
7834     Wrong~line.\\
7835     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
7836     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
7837     number~is~not~valid.~It~will~be~ignored.
7838 }
7839 \@@_msg_new:nn { Impossible~delimiter }
7840 {
7841     Impossible~delimiter.\\
7842     It's~impossible~to~draw~the~#1~delimiter~of~your~
7843     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
7844     in~that~column.

```

```

7845 \bool_if:NT \l_@@_submatrix_slim_bool
7846 { ~Maybe-you-should-try-without-the-key~'slim'. } \\
7847 This~\token_to_str:N \SubMatrix\ will~be~ignored.
7848 }

7849 \@@_msg_new:nn { width~without~X~columns }
7850 {
7851   No~X~column.\\
7852   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column. \\
7853   That~key~will~be~ignored.
7854 }

7855 \@@_msg_new:nn { key~multiplicity~with~dotted }
7856 {
7857   Incompatible~keys. \\
7858   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
7859   in~a~'custom-line'.~They~are~incompatible. \\
7860   The~key~'multiplicity'~will~be~discarded.
7861 }

7862 \@@_msg_new:nn { empty~environment }
7863 {
7864   Empty~environment.\\
7865   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
7866 }

7867 \@@_msg_new:nn { Wrong~use~of~v~center }
7868 {
7869   Wrong~use~of~v~center.\\
7870   You~should~not~use~the~key~'v~center'~here~because~your~block~is~not~
7871   mono~row.~However,~you~can~go~on.
7872 }

7873 \@@_msg_new:nn { No~letter~and~no~command }
7874 {
7875   Erroneous~use.\\
7876   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
7877   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
7878   '~'ccommand'~(to~draw~horizontal~rules).\\
7879   However,~you~can~go~on.
7880 }

7881 \@@_msg_new:nn { Forbidden~letter }
7882 {
7883   Forbidden~letter.\\
7884   You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
7885   It~will~be~ignored.
7886 }

7887 \@@_msg_new:nn { Several~letters }
7888 {
7889   Wrong~name.\\
7890   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
7891   have~used~'\l_@@_letter_str').\\
7892   It~will~be~ignored.
7893 }

7894 \@@_msg_new:nn { Delimiter~with~small }
7895 {
7896   Delimiter~forbidden.\\
7897   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
7898   because~the~key~'small'~is~in~force.\\
7899   This~error~is~fatal.
7900 }

7901 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
7902 {
7903   Unknown~cell.\\
7904   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~

```

```

7905 the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
7906 can't-be-executed~because~a~cell~doesn't~exist.\\
7907 This~command~\token_to_str:N \line\ will~be~ignored.
7908 }
7909 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
7910 {
7911   Duplicate~name.\\
7912   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
7913   in~this~\@@_full_name_env:.\
7914   This~key~will~be~ignored.\\
7915   \bool_if:NF \c_@@_messages_for_Overleaf_bool
7916   { For~a~list~of~the~names~already~used,~type~H~<return>. }
7917 }
7918 {
7919   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
7920   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
7921 }
7922 \@@_msg_new:nn { r-or-l-with-preamble }
7923 {
7924   Erroneous~use.\\
7925   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
7926   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
7927   your~\@@_full_name_env:.\
7928   This~key~will~be~ignored.
7929 }
7930 \@@_msg_new:nn { Hdotsfor~in~col-0 }
7931 {
7932   Erroneous~use.\\
7933   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
7934   the~array.~This~error~is~fatal.
7935 }
7936 \@@_msg_new:nn { bad~corner }
7937 {
7938   Bad~corner.\\
7939   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
7940   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
7941   This~specification~of~corner~will~be~ignored.
7942 }
7943 \@@_msg_new:nn { bad~border }
7944 {
7945   Bad~border.\\
7946   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
7947   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
7948   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
7949   also~use~the~key~'tikz'
7950   \bool_if:NF \c_@@_tikz_loaded_bool
7951   {~if~you~load~the~LaTeX~package~'tikz'}).\
7952   This~specification~of~border~will~be~ignored.
7953 }
7954 \@@_msg_new:nn { tikz~key~without~tikz }
7955 {
7956   Tikz~not~loaded.\\
7957   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
7958   \Block'~because~you~have~not~loaded~Tikz.~
7959   This~key~will~be~ignored.
7960 }
7961 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
7962 {
7963   Erroneous~use.\\
7964   In~the~\@@_full_name_env:,~you~must~use~the~key~
7965   'last~col'~without~value.\\

```

```

7966     However,~you~can~go~on~for~this~time~
7967     (the~value~'\l_keys_value_tl'~will~be~ignored).
7968 }

7969 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
7970 {
7971     Erroneous~use.\\
7972     In~\NiceMatrixoptions,~you~must~use~the~key~
7973     'last-col'~without~value.\\
7974     However,~you~can~go~on~for~this~time~
7975     (the~value~'\l_keys_value_tl'~will~be~ignored).
7976 }

7977 \@@_msg_new:nn { Block~too~large-1 }
7978 {
7979     Block~too~large.\\
7980     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
7981     too~small~for~that~block. \\
7982 }

7983 \@@_msg_new:nn { Block~too~large-2 }
7984 {
7985     Block~too~large.\\
7986     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7987     \g_@@_static_num_of_col_int\
7988     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
7989     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
7990     (&)~at~the~end~of~the~first~row~of~your~
7991     \@@_full_name_env:.\\
7992     This~block~and~maybe~others~will~be~ignored.
7993 }

7994 \@@_msg_new:nn { unknown~column~type }
7995 {
7996     Bad~column~type.\\
7997     The~column~type~'#1'~in~your~\@@_full_name_env:\
7998     is~unknown. \\
7999     This~error~is~fatal.
8000 }

8001 \@@_msg_new:nn { tabularnote~forbidden }
8002 {
8003     Forbidden~command.\\
8004     You~can't~use~the~command~\token_to_str:N\tabularnote\
8005     ~in~a~\@@_full_name_env:~This~command~is~available~only~in~
8006     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
8007     This~command~will~be~ignored.
8008 }

8009 \@@_msg_new:nn { borders~forbidden }
8010 {
8011     Forbidden~key.\\
8012     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
8013     because~the~option~'rounded-corners'~
8014     is~in~force~with~a~non-zero~value.\\
8015     This~key~will~be~ignored.
8016 }

8017 \@@_msg_new:nn { bottomrule~without~booktabs }
8018 {
8019     booktabs~not~loaded.\\
8020     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
8021     loaded~'booktabs'.\\
8022     This~key~will~be~ignored.
8023 }

8024 \@@_msg_new:nn { enumitem~not~loaded }
8025 {

```

```

8026     enumitem~not~loaded.\\
8027     You~can't~use~the~command~\token_to_str:N\tabularnote\
8028     ~because~you~haven't~loaded~'enumitem'.\\
8029     This~command~will~be~ignored.
8030 }
8031 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
8032 {
8033     Tikz~not~loaded.\\
8034     You~have~used~the~key~'tikz'~in~the~definition~of~a~
8035     customized~line~(with~'custom~line')~but~Tikz~is~not~loaded.~
8036     You~can~go~on~but~you~will~have~another~error~if~you~actually~
8037     use~that~custom~line.
8038 }
8039 \@@_msg_new:nn { tikz~in~borders~without~tikz }
8040 {
8041     Tikz~not~loaded.\\
8042     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
8043     command~'\token_to_str:N\Block')~but~Tikz~is~not~loaded.~
8044     That~key~will~be~ignored.
8045 }
8046 \@@_msg_new:nn { color~in~custom~line~with~tikz }
8047 {
8048     Erroneous~use.\\
8049     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
8050     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
8051     The~key~'color'~will~be~discarded.
8052 }
8053 \@@_msg_new:nn { Wrong~last~row }
8054 {
8055     Wrong~number.\\
8056     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
8057     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
8058     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
8059     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
8060     without~value~(more~compilations~might~be~necessary).
8061 }
8062 \@@_msg_new:nn { Yet~in~env }
8063 {
8064     Nested~environments.\\
8065     Environments~of~nicematrix~can't~be~nested.\\
8066     This~error~is~fatal.
8067 }
8068 \@@_msg_new:nn { Outside~math~mode }
8069 {
8070     Outside~math~mode.\\
8071     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
8072     (and~not~in~\token_to_str:N \vcenter).\\
8073     This~error~is~fatal.
8074 }
8075 \@@_msg_new:nn { One~letter~allowed }
8076 {
8077     Bad~name.\\
8078     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
8079     It~will~be~ignored.
8080 }
8081 \@@_msg_new:nn { varwidth~not~loaded }
8082 {
8083     varwidth~not~loaded.\\
8084     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
8085     loaded.\\

```

```

8086     Your~column~will~behave~like~'p'.
8087 }
8088 \@@_msg_new:nnn { Unknown~key~for~Block }
8089 {
8090     Unknown~key.\\
8091     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
8092     \Block.\\ It~will~be~ignored. \\
8093     \c_@@_available_keys_str
8094 }
8095 {
8096     The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
8097     hlines,~hvlines,~l,~line~width,~name,~rounded~corners,~r,~respect~arraystretch,
8098     ~t,~tikz~and~vlines.
8099 }
8100 \@@_msg_new:nn { Version~of~siunitx~too~old }
8101 {
8102     siunitx~too~old.\\
8103     You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
8104     is~too~old.~You~need~at~least~v~3.0~and~your~log~file~says:~"siunitx,~
8105     \use:c { ver @ siunitx.sty }". \\
8106     This~error~is~fatal.
8107 }
8108 \@@_msg_new:nnn { Unknown~key~for~Brace }
8109 {
8110     Unknown~key.\\
8111     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
8112     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
8113     It~will~be~ignored. \\
8114     \c_@@_available_keys_str
8115 }
8116 {
8117     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
8118     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
8119     right~shorten)~and~yshift.
8120 }
8121 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
8122 {
8123     Unknown~key.\\
8124     The~key~'\l_keys_key_str'~is~unknown.\\
8125     It~will~be~ignored. \\
8126     \c_@@_available_keys_str
8127 }
8128 {
8129     The~available~keys~are~(in~alphabetic~order):~
8130     delimiters/color,~
8131     rules~(with~the~subkeys~'color'~and~'width'),~
8132     sub~matrix~(several~subkeys)~
8133     and~xdots~(several~subkeys).~
8134     The~latter~is~for~the~command~\token_to_str:N \line.
8135 }
8136 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
8137 {
8138     Unknown~key.\\
8139     The~key~'\l_keys_key_str'~is~unknown.\\
8140     That~key~will~be~ignored. \\
8141     \c_@@_available_keys_str
8142 }
8143 {
8144     The~available~keys~are~(in~alphabetic~order):~
8145     'delimiters/color',~
8146     'extra~height',~
8147     'hlines',~

```

```

8148 'hvlines',~
8149 'left-xshift',~
8150 'name',~
8151 'right-xshift',~
8152 'rules'~(with~the~subkeys~'color'~and~'width'),~
8153 'slim',~
8154 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
8155 and~'right-xshift').\\
8156 }
8157 \@@_msg_new:nnn { Unknown~key~for~notes }
8158 {
8159   Unknown~key.\\
8160   The~key~'\l_keys_key_str'~is~unknown.\\
8161   That~key~will~be~ignored. \\
8162   \c_@@_available_keys_str
8163 }
8164 {
8165   The~available~keys~are~(in~alphabetic~order):~
8166   bottomrule,~
8167   code-after,~
8168   code-before,~
8169   detect-duplicates,~
8170   enumitem-keys,~
8171   enumitem-keys-para,~
8172   para,~
8173   label-in-list,~
8174   label-in-tabular~and~
8175   style.
8176 }
8177 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
8178 {
8179   Unknown~key.\\
8180   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
8181   \token_to_str:N \RowStyle. \\
8182   That~key~will~be~ignored. \\
8183   \c_@@_available_keys_str
8184 }
8185 {
8186   The~available~keys~are~(in~alphabetic~order):~
8187   'bold',~
8188   'cell-space-top-limit',~
8189   'cell-space-bottom-limit',~
8190   'cell-space-limits',~
8191   'color',~
8192   'nb-rows'~and~
8193   'rowcolor'.
8194 }
8195 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
8196 {
8197   Unknown~key.\\
8198   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
8199   \token_to_str:N \NiceMatrixOptions. \\
8200   That~key~will~be~ignored. \\
8201   \c_@@_available_keys_str
8202 }
8203 {
8204   The~available~keys~are~(in~alphabetic~order):~
8205   allow-duplicate-names,~
8206   cell-space-bottom-limit,~
8207   cell-space-limits,~
8208   cell-space-top-limit,~
8209   code-for-first-col,~
8210   code-for-first-row,~

```

```

8211     code-for-last-col,~
8212     code-for-last-row,~
8213     corners,~
8214     custom-key,~
8215     create-extra-nodes,~
8216     create-medium-nodes,~
8217     create-large-nodes,~
8218     delimiters~(several~subkeys),~
8219     end-of-row,~
8220     first-col,~
8221     first-row,~
8222     hlines,~
8223     hvlines,~
8224     last-col,~
8225     last-row,~
8226     left-margin,~
8227     light-syntax,~
8228     matrix/columns-type,~
8229     notes~(several~subkeys),~
8230     nullify-dots,~
8231     renew-dots,~
8232     renew-matrix,~
8233     respect-arraystretch,~
8234     right-margin,~
8235     rules~(with~the~subkeys~'color'~and~'width'),~
8236     small,~
8237     sub-matrix~(several~subkeys),
8238     vlines,~
8239     xdots~(several~subkeys).
8240 }

8241 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
8242 {
8243     Unknown~key.\\
8244     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
8245     \{NiceArray\}. \\
8246     That~key~will~be~ignored. \\
8247     \c_@@_available_keys_str
8248 }
8249 {
8250     The~available~keys~are~(in~alphabetic~order):~
8251     b,~
8252     baseline,~
8253     c,~
8254     cell-space-bottom-limit,~
8255     cell-space-limits,~
8256     cell-space-top-limit,~
8257     code-after,~
8258     code-for-first-col,~
8259     code-for-first-row,~
8260     code-for-last-col,~
8261     code-for-last-row,~
8262     colortbl-like,~
8263     columns-width,~
8264     corners,~
8265     create-extra-nodes,~
8266     create-medium-nodes,~
8267     create-large-nodes,~
8268     delimiters/color,~
8269     extra-left-margin,~
8270     extra-right-margin,~
8271     first-col,~
8272     first-row,~
8273     hlines,~

```



```

8274     hvlines,~
8275     last-col,~
8276     last-row,~
8277     left-margin,~
8278     light-syntax,~
8279     name,~
8280     notes/bottomrule,~
8281     notes/para,~
8282     nullify-dots,~
8283     renew-dots,~
8284     respect-arraystretch,~
8285     right-margin,~
8286     rules~(with~the~subkeys~'color'~and~'width'),~
8287     small,~
8288     t,~
8289     tabularnote,~
8290     vlines,~
8291     xdots/color,~
8292     xdots/shorten-start,~
8293     xdots/shorten-end,~
8294     xdots/shorten~and~
8295     xdots/line-style.
8296 }

```

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).

```

8297 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
8298 {
8299     Unknown~key.\\
8300     The~key~'\l_keys_key_str'~is~unknown~for~the~
8301     \@@_full_name_env:. \\
8302     That~key~will~be~ignored. \\
8303     \c_@@_available_keys_str
8304 }
8305 {
8306     The~available~keys~are~(in~alphabetic~order):~
8307     b,~
8308     baseline,~
8309     c,~
8310     cell-space-bottom-limit,~
8311     cell-space-limits,~
8312     cell-space-top-limit,~
8313     code-after,~
8314     code-for-first-col,~
8315     code-for-first-row,~
8316     code-for-last-col,~
8317     code-for-last-row,~
8318     colortbl-like,~
8319     columns-type,~
8320     columns-width,~
8321     corners,~
8322     create-extra-nodes,~
8323     create-medium-nodes,~
8324     create-large-nodes,~
8325     delimiters~(several~subkeys),~
8326     extra-left-margin,~
8327     extra-right-margin,~
8328     first-col,~
8329     first-row,~
8330     hlines,~
8331     hvlines,~
8332     l,~
8333     last-col,~
8334     last-row,~

```

```

8335 left-margin,~
8336 light-syntax,~
8337 name,~
8338 nullify-dots,~
8339 r,~
8340 renew-dots,~
8341 respect-arraystretch,~
8342 right-margin,~
8343 rules~(with~the~subkeys~'color'~and~'width'),~
8344 small,~
8345 t,~
8346 vlines,~
8347 xdots/color,~
8348 xdots/shorten-start,~
8349 xdots/shorten-end,~
8350 xdots/shorten-and~
8351 xdots/line-style.
8352 }
8353 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
8354 {
8355   Unknown~key.\\
8356   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
8357   \{NiceTabular\}. \\
8358   That~key~will~be~ignored. \\
8359   \c_@@_available_keys_str
8360 }
8361 {
8362   The~available~keys~are~(in~alphabetic~order):~
8363   b,~
8364   baseline,~
8365   c,~
8366   cell-space-bottom-limit,~
8367   cell-space-limits,~
8368   cell-space-top-limit,~
8369   code-after,~
8370   code-for-first-col,~
8371   code-for-first-row,~
8372   code-for-last-col,~
8373   code-for-last-row,~
8374   colortbl-like,~
8375   columns-width,~
8376   corners,~
8377   custom-line,~
8378   create-extra-nodes,~
8379   create-medium-nodes,~
8380   create-large-nodes,~
8381   extra-left-margin,~
8382   extra-right-margin,~
8383   first-col,~
8384   first-row,~
8385   hlines,~
8386   hvlines,~
8387   last-col,~
8388   last-row,~
8389   left-margin,~
8390   light-syntax,~
8391   name,~
8392   notes/bottomrule,~
8393   notes/para,~
8394   nullify-dots,~
8395   renew-dots,~
8396   respect-arraystretch,~
8397   right-margin,~

```

```

8398 rules~(with~the~subkeys~'color'~and~'width'),~
8399 t,~
8400 tabularnote,~
8401 vlines,~
8402 xdots/color,~
8403 xdots/shorten-start,~
8404 xdots/shorten-end,~
8405 xdots/shorten-and~
8406 xdots/line-style.
8407 }

8408 \@@_msg_new:nnn { Duplicate~name }
8409 {
8410 Duplicate~name.\\
8411 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
8412 the~same~environment~name~twice.~You~can~go~on,~but,~
8413 maybe,~you~will~have~incorrect~results~especially~
8414 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
8415 message~again,~use~the~key~'allow-duplicate-names'~in~
8416 '\token_to_str:N \NiceMatrixOptions'.\\
8417 \c_@@_available_keys_str
8418 }
8419 {
8420 The~names~already~defined~in~this~document~are:~
8421 \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
8422 }

8423 \@@_msg_new:nn { Option~auto~for~columns~width }
8424 {
8425 Erroneous~use.\\
8426 You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
8427 That~key~will~be~ignored.
8428 }

```

20 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 StackExchange⁷⁴, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁷⁵

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`).

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

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`.

⁷⁴cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁷⁵Before 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 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 StackExchange⁷⁶, optimization of Tikz externalization is disabled in the environments of `nicematrix` when the class `standalone` or the package `standalone` is used.

⁷⁶cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

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`.

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 `!\quad` is used in the preamble of the array.

It’s now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

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`.

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`.

Changes between versions 6.3 and 6.4

New commands `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

Correction of a bug of the key `baseline` (cf. question 623258 on TeX StackExchange).

Correction of a bug with the columns `V` of `varwidth`.

Correction of a bug: the use of `\hdottedline` and `:` in the preamble of the array (of another letter specified by `letter-for-dotted-lines`) was incompatible with the key `xdots/line-style`.

Changes between versions 6.4 and 6.5

Key `custom-line` in `\NiceMatrixOptions`.

Key `respect-arraystretch`.

Changes between version 6.5 and 6.6

Keys `tikz` and `width` in `custom-line`.

Changes between version 6.6 and 6.7

Key color for `\OverBrace` and `\UnderBrace` in the `\CodeAfter`
Key `tikz` in the key borders of a command `\Block`

Changes between version 6.7 and 6.8

In the notes of a tabular (with the command `\tabularnote`), the duplicates are now detected: when several commands `\tabularnote` are used with the same argument, only one note is created at the end of the tabular (but all the labels are present, of course).

Changes between version 6.8 and 6.9

New keys `xdots/radius` and `xdots/inter` for customisation of the continuous dotted lines.
New command `\ShowCellNames` available in the `\CodeBefore` and in the `\CodeAfter`.

Changes between version 6.9 and 6.10

New keys `xdots/shorten-start` and `xdots/shorten-end`.
It's possible to use `\line` in the `\CodeAfter` between two blocks (and not only two cells).

Changes between version 6.10 and 6.11

New key `matrix/columns-type` to specify the type of columns of the matrices.
New key `ccommand` in `custom-line` and new command `\cdotteline`.

Changes between version 6.11 and 6.12

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols		
@@ commands:		
<code>\@@_Block:</code>	1290, 5914	
<code>\@@_Block_i</code>	5919, 5920, 5924	
<code>\@@_Block_ii:nnnnn</code>	5924, 5925	
<code>\@@_Block_iv:nnnnn</code>	5962, 5966	
<code>\@@_Block_iv:nnnnnn</code>	6197, 6199	
<code>\@@_Block_v:nnnnn</code>	5963, 6089	
<code>\@@_Block_v:nnnnnn</code>	6226, 6229	
<code>\@@_Cdots</code>	1211, 1282, 4081	
<code>\g_@@_Cdots_lines_tl</code>	1313, 3271	
<code>\@@_CodeAfter:</code>	1294, 6890	
<code>\@@_CodeAfter_i:</code>	935, 2905, 2950, 6891	
<code>\@@_CodeAfter_ii:n</code>	6890, 6891, 6892, 6902	
<code>\@@_CodeAfter_iv:n</code>	6895, 6897	
<code>\@@_CodeAfter_keys:</code>	3212, 3231	
<code>\@@_CodeBefore:w</code>	1456, 1458	
<code>\@@_CodeBefore_Body:w</code>	1378, 1604	
<code>\@@_CodeBefore_keys:</code>	1436, 1453	
<code>\@@_Ddots</code>	1213, 1284, 4113	
<code>\g_@@_Ddots_lines_tl</code>	1316, 3269	
<code>\g_@@_HVdotsfor_lines_tl</code>	1318, 3267, 4197, 4274, 7655	
<code>\@@_Hdotsfor:</code>	1216, 1288, 4173	
<code>\@@_Hdotsfor:nnnn</code>	4199, 4212	
<code>\@@_Hdotsfor_i</code>	4182, 4188, 4195	
<code>\@@_Hline:</code>	1286, 5280	
<code>\@@_Hline_i:n</code>	5280, 5281, 5290	
<code>\@@_Hline_ii:nn</code>	5286, 5290	
<code>\@@_Hline_iii:n</code>	5287, 5291	
<code>\@@_Hspace:</code>	1287, 4167	
<code>\@@_Iddots</code>	1214, 1285, 4137	
<code>\g_@@_Iddots_lines_tl</code>	1317, 3270	
<code>\@@_Ldots</code>	1210, 1215, 1281, 4065	
<code>\g_@@_Ldots_lines_tl</code>	1314, 3272	
<code>\l_@@_Matrix_bool</code>	270, 1728, 1754, 2473, 2843, 2849, 2857, 3019, 6744	
<code>\g_@@_NiceArray_bool</code>	266, 438, 1359, 1599, 1670, 1793, 1800, 1812, 2473, 2831, 2843, 2849, 2857, 2994, 3004, 3036, 3045, 5064, 5068, 5226, 5236, 5266, 5270, 6781, 6794	
<code>\g_@@_NiceMatrixBlock_int</code>	248, 5672, 5677, 5680, 5691	

<code>\l_@@_NiceTabular_bool</code> ...	187, 267, 940, 1128, 1435, 1438, 1566, 1702, 1706, 1801, 1813, 2832, 3059, 3079, 3088, 5996, 6098, 6802
<code>\@@_NotEmpty:</code>	1296, 3051
<code>\@@_OnlyMainNiceMatrix:n</code>	1292, 4840
<code>\@@_OnlyMainNiceMatrix_i:n</code>	4843, 4850, 4853
<code>\@@_OverBrace</code>	3205, 7352
<code>\@@_RowStyle:n</code>	1297, 4433
<code>\@@_S:</code>	236, 1843
<code>\@@_ShowCellNames</code>	1428, 3206, 7504
<code>\@@_SubMatrix</code>	3203, 7048
<code>\@@_SubMatrix_in_code_before</code> ...	1427, 7022
<code>\@@_SubMatrix_in_code_before_i</code> ..	7028, 7031
<code>\@@_SubMatrix_in_code_before_i:nnnn</code> ..	7033, 7034
<code>\@@_UnderBrace</code>	3204, 7347
<code>\@@_V:</code>	1758, 1839
<code>\@@_Vdots</code>	1212, 1283, 4097
<code>\g_@@_Vdots_lines_tl</code>	1315, 3268
<code>\@@_Vdotsfor:</code>	1289, 4272
<code>\@@_Vdotsfor:nnnn</code>	4276, 4287
<code>\@@_W:</code>	1757, 1842, 2345
<code>\@@_X</code>	1851, 3074
<code>\l_@@_X_column_bool</code>	272, 2236, 5960
<code>\l_@@_X_columns_aux_bool</code> ..	304, 1623, 2225
<code>\l_@@_X_columns_dim</code>	305, 1624, 1633, 1639, 2228
<code>\@@_actually_color:</code>	1437, 4525
<code>\@@_actually_diagbox:nnnnnn</code>	5973, 6318, 6814, 6831
<code>\@@_actually_draw_Cdots:</code>	3637, 3641
<code>\@@_actually_draw_Ddots:</code>	3787, 3791
<code>\@@_actually_draw_Iddots:</code>	3835, 3839
<code>\@@_actually_draw_Ldots:</code> ..	3598, 3602, 4263
<code>\@@_actually_draw_Vdots:</code> ..	3719, 3723, 4338
<code>\@@_add_to_colors_seq:nn</code>	4511, 4523, 4524, 4548, 4557, 4566, 4575, 4679
<code>\@@_adjust_pos_of_blocks_seq:</code> ..	3187, 3233
<code>\@@_adjust_pos_of_blocks_seq_i:nnnnn</code> ..	3236, 3238
<code>\@@_adjust_size_box:</code>	1013, 1040, 2087, 2405, 2929, 2974
<code>\@@_adjust_to_submatrix:nn</code>	3482, 3585, 3624, 3705, 3780, 3828
<code>\@@_adjust_to_submatrix:nnnnnn</code> ..	3489, 3491
<code>\@@_after_array:</code>	1739, 3092
<code>\g_@@_after_col_zero_bool</code>	306, 1178, 2906, 4179
<code>\@@_analyze_end:Nn</code>	2624, 2695
<code>\l_@@_argspec_tl</code>	28, 4063, 4064, 4065, 4081, 4097, 4113, 4137, 4193, 4194, 4195, 4270, 4271, 4272, 4354, 4355, 4356, 7046, 7047, 7048
<code>\@@_array:n</code>	1126, 1137, 2627, 2677
<code>\@@_arraycolor</code>	1424, 4610
<code>\c_@@_arydshln_loaded_bool</code>	37, 38
<code>\l_@@_auto_columns_width_bool</code>	562, 727, 2762, 2766, 5667
<code>\g_@@_aux_found_bool</code>	1330, 1335, 1460, 1587, 1590
<code>\g_@@_aux_tl</code>	273, 1593, 1621, 1746, 3101, 3115, 3123, 3220, 5568
<code>\c_@@_available_keys_str</code>	7627, 7629, 7767, 8093, 8114, 8126, 8141, 8162, 8183, 8201, 8247, 8303, 8359, 8417
<code>\l_@@_bar_at_end_of_pream_bool</code>	1794, 1926, 2835
<code>\l_@@_baseline_tl</code>	551, 552, 720, 721, 722, 723, 1135, 1672, 2426, 2438, 2443, 2445, 2450, 2455, 2545, 2546, 2550, 2555, 2557, 2562
<code>\@@_begin_of_NiceMatrix:nn</code>	3017, 3031, 3039, 3048
<code>\@@_begin_of_row:</code>	938, 963, 1241, 2907
<code>\l_@@_block_auto_columns_width_bool</code> ..	1579, 2767, 5660, 5665, 5675, 5685
<code>\g_@@_block_box_int</code>	351, 1560, 5968, 5982, 5984, 6042, 6054, 6066, 6075, 6085
<code>\l_@@_block_name_str</code>	6184, 6265, 6342, 6345, 6350
<code>\@@_block_tikz:nnnnn</code>	6306, 6695
<code>\g_@@_blocks_dp_dim</code>	260, 1021, 1024, 1025, 6069, 6072
<code>\g_@@_blocks_ht_dim</code>	259, 1027, 1030, 1031, 6060, 6063
<code>\g_@@_blocks_seq</code>	321, 1581, 2483, 6079, 6091, 6197
<code>\g_@@_blocks_wd_dim</code>	258, 1015, 1018, 1019, 6048, 6051
<code>\c_@@_booktabs_loaded_bool</code> ..	40, 41, 1230, 2516
<code>\l_@@_borders_clist</code>	339, 6156, 6279, 6603, 6628, 6630, 6632, 6634, 6671, 6690
<code>\l_@@_borders_tikz_tl</code>	6600, 6641, 6657, 6664, 6677, 6684
<code>\@@_brace:nnnnn</code>	7350, 7355, 7371
<code>\l_@@_brace_left_shorten_bool</code>	7359, 7390, 7405
<code>\l_@@_brace_right_shorten_bool</code>	7361, 7411, 7426
<code>\l_@@_brace_yshift_dim</code> ...	7364, 7447, 7479
<code>\@@_c_custom_line:n</code>	5371, 5430
<code>\@@_c_custom_line:nn</code>	5464
<code>\@@_c_custom_line_i:nn</code>	5442, 5447
<code>\@@_cartesian_color:nn</code> ..	4538, 4550, 4559, 4681
<code>\@@_cartesian_path:</code>	4542, 4780
<code>\@@_cartesian_path:n</code>	4590, 4722, 4780
<code>\l_@@_ccommand_str</code>	5312, 5319, 5331, 5371, 5433, 5445
<code>\@@_cell_begin:w</code>	932, 1877, 2007, 2078, 2109, 2235, 2354, 2375, 2396
<code>\l_@@_cell_box</code>	939, 987, 989, 995, 1001, 1004, 1008, 1017, 1018, 1023, 1024, 1029, 1030, 1041, 1042, 1043, 1044, 1046, 1049, 1053, 1055, 1074, 1089, 1091, 1098, 1099, 1112, 1232, 1343, 1345, 2035, 2039, 2043, 2046, 2077, 2088, 2238, 2395, 2406, 2908, 2932, 2935, 2937, 2954, 2977, 2981, 6326, 6447, 6484, 6809
<code>\@@_cell_end:</code>	1034, 1879, 2026, 2083, 2114, 2244, 2356, 2385, 2401
<code>\l_@@_cell_space_bottom_limit_dim</code> ...	533, 637, 1044, 4480
<code>\l_@@_cell_space_top_limit_dim</code>	532, 635, 1042, 4469
<code>\@@_cellcolor</code> ..	1418, 4592, 4604, 4605, 4811
<code>\@@_cellcolor_tabular</code>	1220, 4805

```

\l_@@_cells_seq 2681, 2682, 2687, 2689, 2691
\@@_center_cell_box: ..... 1985, 2030
\@@_chessboardcolors ..... 1426, 4597
\@@_cline ..... 160, 1280
\@@_cline_i:nn ..... 161, 162, 182, 185
\@@_cline_i:w ..... 162, 163
\@@_cline_ii:w ..... 167, 169
\@@_cline_iii:w ..... 166, 170, 171
\l_@@_code_before_bool .....
.. 310, 717, 744, 1151, 1350, 1381, 1596,
2709, 2726, 2744, 2775, 2801, 2838, 2877, 3225
\g_@@_code_before_tl . 1585, 1594, 1597, 3222
\l_@@_code_before_tl .....
..... 309, 716, 1380, 1436, 1597
\l_@@_code_for_first_col_tl ..... 654, 2919
\l_@@_code_for_first_row_tl . 658, 951, 6416
\l_@@_code_for_last_col_tl ..... 656, 2963
\l_@@_code_for_last_row_tl . 660, 958, 6419
\l_@@_code_tl ..... 297, 7017, 7255
\l_@@_col_max_int .....
..... 333, 3348, 3359, 3427, 3487, 3504
\l_@@_col_min_int .....
..... 332, 3353, 3416, 3421, 3485, 3502
\g_@@_col_total_int .....
..... 253, 1057, 1309, 1389, 1400,
1473, 1656, 2326, 2327, 2794, 2795, 2825,
2880, 2885, 2892, 2894, 2953, 3096, 3098,
3110, 3672, 3690, 3750, 4333, 4830, 5718,
5728, 5762, 5849, 6209, 6454, 7081, 7130, 7377
\l_@@_col_width_dim .....
..... 250, 251, 2006, 2076, 6001, 6006
\@@_color:n ..... 212, 217, 1695,
1712, 4490, 5988, 6153, 6502, 6943, 7307, 7330
\l_@@_color_bool .... 5338, 5344, 5379, 5395
\@@_color_index:n ..... 4669, 4690, 4693
\l_@@_color_int .....
..... 4633, 4634, 4651, 4652, 4672, 4683
\l_@@_color_tl ... 341, 4418, 4436, 4485,
4490, 4666, 4667, 4677, 4680, 5909, 5986, 5988
\g_@@_colors_seq 1432, 4514, 4518, 4519, 4529
\l_@@_colors_seq 4628, 4629, 4673, 4692, 4694
\@@_colortbl_like: ..... 1218, 1300
\l_@@_colortbl_like_bool 530, 743, 1300, 1782
\l_@@_colortbl_loaded_bool . 114, 118, 1249
\l_@@_cols_tl .....
..... 4541, 4589, 4620, 4630, 4631, 4681, 4728, 4731
\g_@@_cols_vlism_seq .....
..... 284, 1348, 1777, 1857, 7157
\@@_columncolor ..... 1425, 4553
\@@_columncolor_preamble ..... 1222, 4828
\c_@@_columncolor_regex ..... 73, 1785
\l_@@_columns_type_tl .....
.. 274, 276, 862, 876, 3039, 3048, 6732, 6752
\l_@@_columns_width_dim .....
..... 249, 728, 853, 2763, 2769, 5673, 5679
\g_@@_com_or_env_str ..... 288, 289, 292
\l_@@_command_str .....
..... 5311, 5318, 5329, 5370, 5410, 5427
\@@_computations_for_large_nodes: ..
..... 5789, 5802, 5807
\@@_computations_for_medium_nodes: ..
..... 5709, 5778, 5788, 5799
\@@_compute_a_corner:nnnnnn .....
..... 5556, 5558, 5560, 5562, 5575
\@@_compute_corners: ..... 3186, 5548
\@@_compute_i_j:nn ..... 7057, 7078, 7374
\@@_compute_i_j:nnnn ..... 7059, 7060
\@@_compute_rule_width:n .....
..... 5414, 5438, 5465, 5489
\l_@@_corners_cells_seq .....
..... 325, 4725, 4765, 4940, 4946, 4953,
5128, 5134, 5141, 5550, 5566, 5570, 5571, 5631
\l_@@_corners_clist 556, 710, 4914, 5102, 5551
\@@_create_blocks_nodes: ..... 1409, 1505
\@@_create_col_nodes: .... 2632, 2648, 2701
\@@_create_diag_nodes: ... 1406, 3131, 3289
\@@_create_extra_nodes: .. 1503, 2482, 5699
\@@_create_large_nodes: ..... 5707, 5783
\@@_create_medium_and_large_nodes: ..
..... 5704, 5794
\@@_create_medium_nodes: ..... 5705, 5773
\@@_create_nodes: 5780, 5791, 5801, 5804, 5845
\@@_create_one_block_node:nnnnn 1511, 1514
\@@_create_row_node: ..... 1139, 1181
\@@_create_row_node_i: ... 1144, 1147, 1231
\@@_custom_line:n ..... 628, 5309, 5502
\l_@@_custom_line_commands_seq .....
..... 301, 1298, 5427, 5445
\@@_custom_line_i:n ..... 5322, 5334
\@@_cut_on_hyphen:w .....
..... 371, 384, 4582, 4587, 4647, 4735,
4736, 4758, 4759, 4789, 4790, 5450, 5451,
6510, 6519, 6550, 6553, 6573, 6576, 6604, 6607
\g_@@_ddots_int ..... 3163, 3807, 3808
\@@_def_env:nnn .....
..... 3000, 3012, 3013, 3014, 3015, 3016
\@@_define_com:nnn .....
..... 6777, 6786, 6787, 6788, 6789, 6790
\@@_delimiter:nnn .....
... 2142, 2163, 2171, 2184, 2190, 2196, 6905
\l_@@_delimiters_color_tl 575, 791, 1449,
1695, 1712, 6884, 6943, 6970, 6992, 7307, 7330
\l_@@_delimiters_max_width_bool .....
..... 576, 789, 1716
\g_@@_delta_x_one_dim .... 3165, 3810, 3820
\g_@@_delta_x_two_dim .... 3167, 3858, 3868
\g_@@_delta_y_one_dim .... 3166, 3812, 3820
\g_@@_delta_y_two_dim .... 3168, 3860, 3868
\@@_diagbox:nn ..... 1295, 6810
\@@_dotfill: ..... 6799
\@@_dotfill_i: ..... 6804, 6806
\@@_dotfill_ii: ..... 6803, 6806, 6807
\@@_dotfill_iii: ..... 6807, 6808
\l_@@_dotted_bool .....
.... 359, 3880, 4875, 4962, 4964, 5150, 5152
\l_@@_dotted_rule_bool .....
... 5337, 5350, 5383, 5393, 5406, 5469, 5473
\@@_double_int_eval:n .... 4344, 4364, 4365
\@@_double_int_eval_i:n ..... 4348, 4350
\g_@@_dp_ante_last_row_dim ..... 966, 1265
\g_@@_dp_last_row_dim .....
.... 966, 967, 1268, 1269, 1344, 1345, 1689
\g_@@_dp_row_zero_dim .....
.... 986, 987, 1259, 1260, 1682, 2539, 2578
\@@_draw_Cdots:nnn ..... 3622

```

<code>\@@_draw_Ddots:nnn</code>	3778	4117, 4118, 4123, 4124, 4141, 4142, 4147,
<code>\@@_draw_Iddots:nnn</code>	3826	4148, 5564, 7007, 7082, 7112, 7115, 7380, 7381
<code>\@@_draw_Ldots:nnn</code>	3583	<code>\@@_error:nnn</code>
<code>\@@_draw_Vdots:nnn</code>	3703	14, 4381, 7186, 7221
<code>\@@_draw_blocks:</code>	2483, 6194	<code>\@@_error_too_much_cols:</code>
<code>\@@_draw_dotted_lines:</code>	3185, 3256	1822, 7640
<code>\@@_draw_dotted_lines_i:</code>	3259, 3263	<code>\@@_everycr:</code>
<code>\l_@@_draw_first_bool</code>	348, 4128, 4152, 4163	1174, 1254, 1257
<code>\@@_draw_hlines:</code>	3188, 5262	<code>\@@_everycr_i:</code>
<code>\@@_draw_line:</code>	3620,	1174, 1175
3665, 3776, 3824, 3872, 3874, 4403, 5040, 5242		<code>\l_@@_except_borders_bool</code>
<code>\@@_draw_line_ii:nn</code>	4383, 4387	567, 682, 5064, 5068, 5266, 5270
<code>\@@_draw_line_iii:nn</code>	4390, 4394	<code>\@@_exec_code_before:</code>
<code>\@@_draw_standard_dotted_line:</code>	3881, 3913	1350, 1430
<code>\@@_draw_standard_dotted_line_i:</code>	3976, 3980	<code>\@@_exp_color_arg:Nn</code>
<code>\l_@@_draw_tl</code>	337, 6148,	205, 211, 215, 4443, 4451
6154, 6267, 6492, 6498, 6500, 6502, 6539, 6540		<code>\@@_expand_clist:N</code>
<code>\@@_draw_unstandard_dotted_line:</code>	3882, 3884	376, 1228, 1229
<code>\@@_draw_unstandard_dotted_line:n</code>	3887, 3890, 3897	<code>\@@_expand_clist:NN</code>
<code>\@@_draw_unstandard_dotted_line:nnn</code>	3892, 3898, 3912	4728, 4729, 4781
<code>\@@_draw_vlines:</code>	3189, 5060	<code>\l_@@_exterior_arraycolsep_bool</code>
<code>\g_@@_empty_cell_bool</code>	318, 1048, 1058,	553, 849, 1803, 1815, 2834
2942, 2989, 4079, 4095, 4111, 4135, 4158, 4169		<code>\l_@@_extra_left_margin_dim</code>
<code>\l_@@_end_int</code>	4868, 4869,	570, 698, 1372, 2940
4892, 4893, 4904, 4931, 5081, 5082, 5092, 5119		<code>\l_@@_extra_right_margin_dim</code>
<code>\l_@@_end_of_row_tl</code>	572, 573, 648, 2655, 2656, 7812	571, 699, 1612, 2985, 3753
<code>\c_@@_endpgfortikzpicture_tl</code>	53, 58, 3260, 4391, 6619	<code>\@@_extract_coords_values:</code>
<code>\c_@@_enumitem_loaded_bool</code>	43, 44, 411, 769, 778	5870, 5877
<code>\@@_env:</code>	243, 247, 972, 978,	<code>\@@_fatal:n</code>
1075, 1081, 1095, 1103, 1157, 1163, 1169,		15, 280, 708, 1569, 2119, 2123, 2152,
1244, 1390, 1391, 1396, 1397, 1402, 1403,		2637, 2641, 2643, 2698, 4184, 7645, 7648, 7651
1415, 1470, 1471, 1476, 1479, 1482, 1485,		<code>\@@_fatal:nn</code>
1494, 1495, 1500, 1501, 1527, 2710, 2713,		16, 1869, 2348
2715, 2731, 2737, 2740, 2749, 2755, 2758,		<code>\l_@@_fill_tl</code>
2780, 2786, 2789, 2807, 2813, 2819, 2846,		336, 6146, 6289, 6294, 6295, 6296
2855, 2869, 2880, 2885, 2894, 3136, 3137,		<code>\l_@@_final_i_int</code>
3142, 3143, 3151, 3157, 3198, 3306, 3308,		3172, 3335, 3340, 3343, 3368,
3315, 3317, 3318, 3323, 3326, 3388, 3456,		3376, 3380, 3389, 3397, 3477, 3533, 3614,
3521, 3532, 3545, 3548, 3567, 3570, 3675,		3687, 3693, 3696, 4217, 4245, 4313, 4323, 4325
3678, 3693, 3696, 4226, 4244, 4301, 4319,		<code>\l_@@_final_j_int</code>
4376, 4378, 4397, 4400, 5586, 5605, 5623,		3173, 3336, 3341, 3348, 3353, 3359, 3369,
5731, 5733, 5741, 5852, 5861, 5879, 6340,		3377, 3381, 3390, 3398, 3476, 3478, 3534,
6345, 6346, 6351, 6360, 6364, 6378, 6383,		3567, 3570, 3578, 4238, 4248, 4250, 4292, 4321
6395, 6401, 6402, 6405, 6424, 6461, 6920,		<code>\l_@@_final_open_bool</code>
6923, 7097, 7099, 7104, 7106, 7133, 7135,		3175, 3342, 3346, 3349, 3356,
7140, 7142, 7239, 7251, 7396, 7398, 7417, 7419		3362, 3366, 3382, 3611, 3646, 3651, 3662,
<code>\g_@@_env_int</code>	242, 243, 245, 1578, 1588, 1591, 1745, 5888	3726, 3736, 3741, 3762, 3799, 3847, 3984,
<code>\@@_error:n</code>	11,	3999, 4215, 4239, 4251, 4290, 4314, 4326, 4373
414, 439, 585, 616, 624, 670, 785, 843, 852,		<code>\@@_find_extremities_of_line:nnnn</code>
864, 882, 889, 897, 898, 899, 905, 910, 911,		3330, 3588, 3627, 3708, 3783, 3831
912, 926, 928, 929, 930, 1451, 1617, 1651,		<code>\l_@@_first_col_int</code>
1661, 1732, 2067, 2460, 2521, 2567, 4431,		148, 161, 362, 363, 650, 903, 938,
4623, 5321, 5343, 5345, 5352, 5356, 5360,		1400, 1473, 1665, 1795, 2674, 2704, 2724,
5391, 6144, 6192, 6237, 6598, 6642, 6648,		3108, 3672, 3690, 4177, 4749, 4842, 5718,
6888, 7002, 7013, 7020, 7369, 7584, 7611, 7621		5728, 5762, 5810, 5849, 6758, 6764, 6770, 7130
<code>\@@_error:nn</code>	12, 13, 735, 2145, 2191, 2197,	<code>\l_@@_first_i_tl</code>
2221, 4068, 4071, 4084, 4087, 4100, 4103,		7062,
		7066, 7067, 7093, 7124, 7133, 7135, 7189,
		7196, 7198, 7280, 7291, 7295, 7393, 7414, 7442
		<code>\l_@@_first_j_tl</code>
		7063, 7068, 7069, 7097, 7099, 7159, 7172,
		7179, 7181, 7281, 7292, 7296, 7396, 7398, 7408
		<code>\l_@@_first_row_int</code>
		360, 361, 651, 907,
		1307, 1394, 1468, 1680, 2457, 2536, 2564,
		2575, 3105, 3542, 3564, 5711, 5725, 5752,
		5809, 5847, 6357, 6375, 6756, 6917, 7094, 7819
		<code>\c_@@_footnote_bool</code>
		1555, 1749, 7560, 7582, 7605, 7608, 7618, 7624
		<code>\c_@@_footnotehyper_bool</code>
		7559, 7583, 7615
		<code>\c_@@_forbidden_letters_str</code>
		5359, 5373
		<code>\@@_full_name_env:</code>
		290,
		7669, 7676, 7685, 7695, 7710, 7828, 7836,
		7865, 7897, 7905, 7913, 7919, 7925, 7927,
		7964, 7986, 7991, 7997, 8005, 8057, 8071, 8301
		<code>\@@_h_custom_line:n</code>
		5370, 5408

\@@_h_custom_line:nn	5429	5831, 5835, 5836, 5849, 5852, 5853, 5855,
\@@_hline:n	5078, 5277, 5300, 5418, 5454, 6583	5860, 5861, 5873, 5879, 5880, 5882, 5887, 5888
\@@_hline_i:	5084, 5087	\l_@@_key_nb_rows_int
\@@_hline_ii:	5112, 5120, 5148	255, 4425, 4426, 4437, 4447, 4454, 4461
\@@_hline_iii:	5156, 5160	\l_@@_l_dim
\@@_hline_iv:	5153, 5213	3960, 3961, 3974, 3975, 3987, 3993,
\@@_hline_v:	5157, 5245	4004, 4013, 4023, 4028, 4035, 4038, 4045, 4048
\@@_hlines_block:nnn	6255, 6569	\l_@@_large_nodes_bool
\l_@@_hlines_block_bool	350, 6161, 6251, 6262	566, 689, 5703, 5707
\l_@@_hlines_clist	355, 662, 676,	\g_@@_last_col_found_bool
	681, 1182, 1184, 1188, 1228, 3188, 5275, 5276	370,
\l_@@_hpos_block_str	343, 344, 5893,	1312, 1657, 1723, 2793, 2872, 2951, 3095, 6452
	5895, 5897, 5899, 5901, 5903, 5940, 5941,	\l_@@_last_col_int
	5943, 5995, 6007, 6020, 6031, 6082, 6106,	368, 369,
	6110, 6123, 6128, 6165, 6167, 6169, 6171,	844, 873, 875, 890, 906, 927, 1333, 1336,
	6174, 6177, 6428, 6440, 6451, 6455, 6465, 6477	1660, 1807, 2671, 2673, 3024, 3026, 3096,
\l_@@_hpos_cell_str	256, 257,	3098, 3713, 3748, 4070, 4086, 4124, 4148,
	1877, 1973, 1975, 2079, 2354, 2397, 5939, 5941	6202, 6207, 6208, 6209, 6212, 6248, 6258,
\l_@@_hpos_col_str		6274, 6286, 6298, 6310, 6322, 6337, 6378,
	1929, 1931, 1933, 1935, 1960, 1972,	6383, 6391, 6407, 6760, 6766, 6772, 7644, 7677
	1976, 1978, 1986, 1987, 1990, 1995, 2062, 2213	\l_@@_last_col_without_value_bool
\l_@@_hpos_of_block_cap_bool		367, 872, 3097, 7647
	345, 6172, 6175, 6178, 6354, 6425, 6462	\l_@@_last_empty_column_int
\g_@@_ht_last_row_dim		5596, 5597, 5610, 5616, 5629
	968, 1266, 1267, 1342, 1343, 1688	\l_@@_last_empty_row_int
\g_@@_ht_row_one_dim	994, 995, 1263, 1264	5578, 5579, 5592, 5613
\g_@@_ht_row_zero_dim		\l_@@_last_i_tl
	988, 989, 1261, 1262, 1683, 2538, 2577	7064, 7070, 7071, 7080, 7093, 7127,
\@@_i:	5711, 5713,	7140, 7142, 7189, 7196, 7376, 7393, 7414, 7474
	5714, 5715, 5716, 5725, 5731, 5733, 5734,	\l_@@_last_j_tl
	5735, 5736, 5741, 5742, 5743, 5744, 5752,	7065, 7072, 7073, 7081, 7104,
	5755, 5757, 5758, 5759, 5811, 5813, 5816,	7106, 7162, 7172, 7179, 7377, 7417, 7419, 7429
	5817, 5821, 5822, 5847, 5852, 5854, 5856,	\l_@@_last_row_int
	5860, 5861, 5872, 5879, 5881, 5883, 5887, 5888	364,
\g_@@_iddots_int	3164, 3855, 3856	365, 652, 956, 1002, 1194, 1327, 1331,
\l_@@_in_env_bool	263, 438, 1569, 1570	1338, 1645, 1649, 1652, 1664, 1686, 2660,
\l_@@_initial_i_int	3170,	2661, 2915, 2916, 2960, 2961, 3100, 3593,
	3333, 3408, 3411, 3436, 3444, 3448, 3457,	3632, 4102, 4118, 4142, 4848, 4856, 6201,
	3465, 3475, 3522, 3607, 3653, 3655, 3669,	6204, 6205, 6224, 6248, 6258, 6274, 6286,
	3675, 3678, 4216, 4217, 4227, 4295, 4305, 4307	6298, 6309, 6321, 6335, 6406, 6418, 6768, 8056
\l_@@_initial_j_int		\g_@@_last_row_node_int
	3171, 3334, 3409, 3416,	254, 1141, 1143, 1354
	3421, 3427, 3437, 3445, 3449, 3458, 3466,	\l_@@_last_row_without_value_bool
	3476, 3478, 3523, 3545, 3548, 3556, 3743,	366, 1329, 1647, 3099
	3745, 3750, 4220, 4230, 4232, 4291, 4292, 4303	\l_@@_left_delim_dim
\l_@@_initial_open_bool	3174, 3410, 3414,	1357, 1361, 1366, 2613, 2938
	3417, 3424, 3430, 3434, 3450, 3604, 3643,	\g_@@_left_delim_tl
	3650, 3660, 3726, 3733, 3739, 3793, 3841,	1365, 1557, 1696, 1719, 1791, 2126, 2128, 5228
	3982, 4214, 4221, 4233, 4289, 4296, 4308, 4372	\l_@@_left_margin_dim
\@@_insert_tabulartnotes:	2488, 2491	568, 692, 1371, 2939, 5225, 5840
\@@_instruction_of_type:nnn		\l_@@_letter_str
	1115, 4073, 4089, 4105, 4128, 4152	5313,
\c_@@_integers_alist_tl	7265, 7276	5317, 5327, 5353, 5355, 5359, 5364, 7884, 7891
\g_@@_internal_code_after_tl		\l_@@_letter_vlism_tl
	298, 1913, 2141,	283, 669, 1855
	2162, 2170, 2183, 2189, 2195, 3208, 3209,	\l_@@_light_syntax_bool
	5298, 5416, 5452, 5492, 5971, 6316, 6812, 7026	550, 646, 1374, 1607
\@@_intersect_our_row:nnnnn	4711	\@@_light_syntax_i:w
\@@_intersect_our_row_p:nnnnn	4661	2645, 2651
\@@_j:	5718, 5720,	\@@_line
	5721, 5722, 5723, 5728, 5731, 5733, 5736,	3207, 4356
	5738, 5739, 5741, 5744, 5746, 5747, 5762,	\@@_line_i:nn
	5765, 5767, 5768, 5769, 5824, 5826, 5829,	4363, 4370
		\l_@@_line_width_dim
		342, 6163, 6493, 6531, 6542, 6548, 6571,
		6595, 6627, 6653, 6655, 6672, 6673, 6675, 6693
		\@@_line_with_light_syntax:n
		2665, 2669, 2679, 2694
		\l_@@_local_end_int
		4902, 4923, 4931, 4988, 5037,
		5052, 5090, 5111, 5119, 5176, 5231, 5233, 5254
		\l_@@_local_start_int
		4901, 4917, 4918, 4921,

4925, 4929, 4977, 5035, 5048, 5089, 5105,
 5106, 5109, 5113, 5117, 5165, 5221, 5223, 5250
 \@@_math_toggle_token: 186, 1036,
 2909, 2926, 2955, 2971, 6857, 6868, 7460, 7496
 \g_@@_max_cell_width_dim
 1045, 1046, 1580, 2768, 5666, 5692
 \c_@@_max_l_dim 3974, 3979
 \l_@@_medium_nodes_bool 565, 688, 5701, 6398
 \@@_message_hdotsfor: 7653, 7676, 7685, 7695
 \c_@@_messages_for_Overleaf_bool
 20, 25, 7581, 7626, 7915
 \@@_msg_new:nn 17, 7587, 7596,
 7658, 7665, 7671, 7680, 7690, 7700, 7707,
 7715, 7721, 7727, 7733, 7738, 7744, 7750,
 7756, 7779, 7787, 7794, 7801, 7807, 7815,
 7824, 7832, 7839, 7849, 7855, 7862, 7867,
 7873, 7881, 7887, 7894, 7901, 7922, 7930,
 7936, 7943, 7954, 7961, 7969, 7977, 7983,
 7994, 8001, 8009, 8017, 8024, 8031, 8039,
 8046, 8053, 8062, 8068, 8075, 8081, 8100, 8423
 \@@_msg_new:nnn
 18, 7762, 7909, 8088, 8108, 8121,
 8136, 8157, 8177, 8195, 8241, 8297, 8353, 8408
 \@@_msg_redirect_name:nn
 26, 855, 1735, 6215, 6218
 \@@_multicolumn:nnn 1302, 2286
 \g_@@_multicolumn_cells_seq
 328, 1305, 1352, 2301,
 3121, 3125, 3126, 5736, 5744, 5866, 6362, 6380
 \g_@@_multicolumn_sizes_seq
 329, 1306, 1353, 2303, 3127, 3128, 5867
 \l_@@_multiplicity_int 4873,
 4985, 4986, 4992, 5004, 5014, 5173, 5174,
 5180, 5190, 5200, 5349, 5376, 5398, 5480, 5481
 \g_@@_name_env_str 287, 293, 294,
 1564, 1565, 2697, 2995, 2996, 3005, 3006,
 3037, 3046, 3057, 3075, 3085, 3227, 6782, 7642
 \l_@@_name_str
 564, 737, 974, 977, 1077, 1080, 1165,
 1168, 2714, 2715, 2739, 2740, 2757, 2758,
 2788, 2789, 2815, 2818, 2865, 2868, 2887,
 2891, 3145, 3150, 3156, 3307, 3308, 3319,
 3322, 3325, 5857, 5860, 5884, 5887, 6347, 6350
 \g_@@_names_seq 262, 734, 736, 8421
 \l_@@_nb_cols_int 2663, 2674,
 2683, 2686, 6728, 6741, 6751, 6759, 6765, 6771
 \l_@@_nb_rows_int 6727, 6740, 6762
 \l_@@_new_body_tl 2662, 2668, 2677, 2690, 2692
 \@@_newcolumnntype 1201, 1756, 1757, 1758, 3063
 \@@_node_for_cell:
 1054, 1061, 1441, 2936, 2986
 \@@_node_for_multicolumn:nn 5868, 5875
 \@@_node_left:nn 7238, 7239, 7299
 \@@_node_position:
 1396, 1398, 1402, 1404, 1470, 1472, 1480, 1486
 \@@_node_position_i: 1483, 1487
 \@@_node_right:nnnn 7248, 7250, 7322
 \g_@@_not_empty_cell_bool
 308, 1052, 1059, 3052
 \@@_not_in_exterior:nnnnn 4703
 \@@_not_in_exterior_p:nnnnn 4639
 \l_@@_notes_above_space_dim 557, 559
 \l_@@_notes_bottomrule_bool
 755, 893, 921, 2514
 \l_@@_notes_code_after_tl 753, 2523
 \l_@@_notes_code_before_tl 751, 2495
 \l_@@_notes_detect_duplicates_bool
 264, 265, 442, 783
 \@@_notes_format:n 400, 454, 459
 \@@_notes_label_in_list:n 407, 426, 434, 763
 \@@_notes_label_in_tabular:n . 406, 465, 760
 \l_@@_notes_labels_seq 398, 453, 458, 468, 473
 \l_@@_notes_para_bool . . 749, 891, 919, 2499
 \@@_notes_style:n 403, 405, 408, 426, 434, 757
 \l_@@_nullify_dots_bool
 560, 687, 4077, 4093, 4109, 4133, 4156
 \@@_old_CT@arc@ 1571, 3229
 \@@_old_cdots 1274, 4094
 \@@_old_ddots 1276, 4134
 \@@_old_dotfill 6798, 6801, 6809
 \@@_old_dotfill: 1293
 \l_@@_old_iRow_int 299, 1323, 3276
 \@@_old_ialign: 1138, 1270, 3202, 6196
 \@@_old_iddots 1277, 4157
 \l_@@_old_jCol_int 300, 1325, 3277
 \@@_old_ldots 1273, 4078
 \@@_old_multicolumn 1304, 4172
 \@@_old_pgfpaintanchor 218, 7257, 7261
 \@@_old_pgful@check@rerun 107, 111
 \@@_old_vdots 1275, 4110
 \@@_open_x_final_dim:
 3561, 3613, 3647, 3801, 3850
 \@@_open_x_initial_dim:
 3539, 3606, 3644, 3796, 3844
 \@@_open_y_final_dim: 3685, 3737, 3849
 \@@_open_y_initial_dim:
 3667, 3734, 3795, 3843
 \l_@@_other_keys_tl
 4894, 4963, 5083, 5151, 5314, 5322
 \@@_overbrace_i:n 7435, 7440
 \l_@@_parallelize_diags_bool
 554, 555, 684, 3161, 3805, 3853
 \@@_patch_for_revtext: 1534, 1553
 \@@_patch_m_preamble:n
 2283, 2295, 2330, 2363, 2368, 2415
 \@@_patch_m_preamble_i:n
 2334, 2335, 2336, 2350
 \@@_patch_m_preamble_ii:nn
 2337, 2338, 2339, 2360
 \@@_patch_m_preamble_iii:n 2340, 2365
 \@@_patch_m_preamble_iv:nnn
 2341, 2342, 2343, 2370
 \@@_patch_m_preamble_ix:n 2414, 2417
 \@@_patch_m_preamble_v:nnnn 2344, 2345, 2390
 \@@_patch_m_preamble_x:n
 2358, 2388, 2409, 2411, 2420
 \@@_patch_node_for_cell: 1087, 1441
 \@@_patch_node_for_cell:n 1085, 1111, 1114
 \@@_patch_preamble:n
 1779, 1825, 1861, 1867, 1887,
 1923, 2130, 2146, 2148, 2164, 2172, 2198, 2269
 \@@_patch_preamble_i:n 1829, 1830, 1831, 1873
 \@@_patch_preamble_ii:nn
 1832, 1833, 1834, 1884
 \@@_patch_preamble_iii:n . 1835, 1889, 1897


```

\@@_patch_preamble_iii_i:n .... 1892, 1894
\@@_patch_preamble_iv:n .....
..... 1836, 1837, 1838, 1945
\@@_patch_preamble_iv_i:n .... 1948, 1950
\@@_patch_preamble_iv_ii:w 1953, 1954, 1956
\@@_patch_preamble_iv_iii:nn ... 1957, 1958
\@@_patch_preamble_iv_iv:nn .....
..... 1962, 1964, 2065, 2068, 2227
\@@_patch_preamble_iv_v:nnnnnnnn 1968, 2001
\@@_patch_preamble_ix:nn .....
..... 1847, 1848, 1849, 2150
\@@_patch_preamble_ix_i:nnn ... 2154, 2176
\@@_patch_preamble_v:n ... 1839, 1840, 2051
\@@_patch_preamble_v_i:w . 2054, 2055, 2057
\@@_patch_preamble_v_ii:nn .... 2058, 2059
\@@_patch_preamble_vi:nnnn 1841, 1842, 2071
\@@_patch_preamble_vii:n ..... 1843, 2094
\@@_patch_preamble_vii_i:w 2097, 2098, 2100
\@@_patch_preamble_vii_ii:n .... 2101, 2102
\@@_patch_preamble_viii:nn .....
..... 1844, 1845, 1846, 2121
\@@_patch_preamble_viii_i:nn .....
..... 2134, 2137, 2139
\@@_patch_preamble_x:n ... 1850, 1851, 2201
\@@_patch_preamble_x_i:w . 2204, 2205, 2207
\@@_patch_preamble_x_ii:n ..... 2208, 2211
\@@_patch_preamble_xi:n .....
... 1882, 1999, 2092, 2117, 2248, 2251, 2275
\@@_patch_preamble_xiii:n ..... 2254, 2272
\@@_pgf_rect_node:nnn ..... 506, 1484, 6400
\@@_pgf_rect_node:nnnnn .....
..... 481, 1526, 5851, 5878, 6339, 6394, 7225
\c_@@_pgfortikzpicture_tl .....
..... 52, 57, 3258, 4389, 6617
\@@_pgfpntanchor:n ..... 7253, 7258
\@@_pgfpntanchor_i:nn ..... 7261, 7263
\@@_pgfpntanchor_ii:w ..... 7264, 7272
\@@_pgfpntanchor_iii:w ..... 7285, 7287
\@@_picture_position: .....
..... 1391, 1398, 1404, 1472, 1486, 1487
\g_@@_pos_of_blocks_seq .....
..... 322, 1351, 1510, 1582, 2304, 3113, 3117,
..... 3118, 3235, 4637, 4908, 5096, 5643, 6264, 6822
\g_@@_pos_of_stroken_blocks_seq .....
..... 324, 1583, 4912, 5100, 6276
\g_@@_pos_of_xdots_seq .....
..... 323, 1584, 3473, 4910, 5098
\l_@@_position_int ..... 4860, 4895,
..... 4903, 4979, 5032, 5050, 5091, 5167, 5218, 5252
\g_@@_post_action_cell_tl .....
..... 934, 1038, 2032, 2237, 4467, 4478
\@@_pre_array: ..... 1321, 1382, 1604
\@@_pre_array_ii: ..... 1224, 1355
\@@_pre_code_before: ..... 1384, 1462
\c_@@_preamble_first_col_tl .... 1796, 2901
\c_@@_preamble_last_col_tl .... 1808, 2946
\g_@@_preamble_tl 1559, 1759, 1763, 1767,
..... 1773, 1787, 1796, 1805, 1808, 1817, 1821,
..... 1859, 1875, 1886, 1899, 2003, 2073, 2106,
..... 2133, 2161, 2169, 2182, 2188, 2232, 2258,
..... 2265, 2274, 2282, 2284, 2294, 2296, 2352,
..... 2362, 2367, 2372, 2392, 2419, 2627, 2677, 5490
\@@_provide_pgfsyspdfmark: ... 74, 83, 1554

\@@_put_box_in_flow: ..... 1721, 2422, 2615
\@@_put_box_in_flow_bis:nn .... 1718, 2582
\@@_put_box_in_flow_i: ..... 2428, 2430
\@@_qpoint:n .....
..... 246, 1518, 1520, 1522, 1524, 2433, 2435,
..... 2447, 2463, 2530, 2532, 2548, 2559, 2570,
..... 3295, 3297, 3299, 3301, 3311, 3313, 3556,
..... 3578, 3607, 3614, 3653, 3655, 3669, 3687,
..... 3743, 3745, 4397, 4400, 4748, 4752, 4768,
..... 4770, 4977, 4979, 4988, 5032, 5035, 5037,
..... 5048, 5050, 5052, 5165, 5167, 5176, 5218,
..... 5221, 5231, 5250, 5252, 5254, 5757, 5767,
..... 6331, 6333, 6335, 6337, 6371, 6391, 6421,
..... 6515, 6517, 6524, 6526, 6652, 6654, 6656,
..... 6670, 6674, 6676, 6836, 6838, 6841, 6843,
..... 6910, 6912, 7124, 7127, 7164, 7181, 7198,
..... 7408, 7429, 7442, 7474, 7512, 7514, 7523, 7525
\l_@@_real_left_delim_dim 2584, 2599, 2614
\l_@@_real_right_delim_dim 2585, 2611, 2617
\@@_recreate_cell_nodes: ..... 1407, 1466
\g_@@_recreate_cell_nodes_bool .....
..... 563, 1239, 1407, 1433, 1440, 1445
\@@_rectanglecolor .....
..... 1419, 4443, 4562, 4595, 4612, 4822
\@@_rectanglecolor:nnn ... 4568, 4577, 4580
\@@_renew_NC@rewrite@S: .... 229, 231, 1311
\@@_renew_dots: ..... 1208, 1301
\l_@@_renew_dots_bool ..... 685, 1301, 7577
\@@_renew_matrix: ..... 847, 6707, 7579
\l_@@_respect_arraystretch_bool .....
..... 561, 703, 5911, 5991, 6002, 6101, 6118, 6187
\l_@@_respect_blocks_bool 4618, 4635, 4658
\@@_restore_iRow_jCol: ..... 3228, 3274
\c_@@_revtex_bool ..... 62, 65, 68, 69, 1553
\l_@@_right_delim_dim .....
..... 1358, 1362, 1368, 2616, 2983
\g_@@_right_delim_tl . 1367, 1558, 1713,
..... 1719, 1792, 2129, 2158, 2159, 2180, 2185, 5238
\l_@@_right_margin_dim .....
..... 569, 694, 1611, 2984, 3752, 5235, 5843
\@@_rotate: ..... 1291, 4343
\g_@@_rotate_bool .....
..... 271, 1011, 1039, 2023, 2086, 2404, 2928,
..... 2973, 4343, 5995, 6039, 6044, 6105, 6122, 6327
\@@_rotate_cell_box: .....
..... 999, 1039, 2086, 2404, 2928, 2973, 6327
\l_@@_rounded_corners_dim .....
..... 340, 6150, 6299, 6507, 6508, 6543, 6597, 6691
\@@_roundedrectanglecolor ..... 1420, 4571
\l_@@_row_max_int .... 331, 3343, 3486, 3503
\l_@@_row_min_int .... 330, 3411, 3484, 3501
\g_@@_row_of_col_done_bool .....
..... 307, 1179, 1563, 2723
\g_@@_row_style_tl .....
..... 311, 946, 1586, 2015, 4459, 4460,
..... 4462, 4465, 4476, 4487, 4495, 4506, 4508, 5989
\g_@@_row_total_int .... 252, 1308, 1388,
..... 1394, 1468, 1663, 2458, 2565, 3100, 3107,
..... 3542, 3564, 4258, 5711, 5725, 5752, 5847,
..... 6224, 6357, 6375, 6917, 7080, 7094, 7376, 7820
\@@_rowcolor ..... 1421, 4451, 4544
\@@_rowcolor_tabular ..... 1221, 4816
\@@_rowcolors ..... 1422, 4696

```

\@@_rowcolors_i:nnnnn	4662, 4698	\l_@@_tabularnote_tl	397, 895, 923, 2487, 2496
\l_@@_rowcolors_restart_bool ...	4621, 4650	\g_@@_tabularnotes_seq	396, 444, 455, 2486, 2502, 2508, 2524
\@@_rowlistcolors	1423, 4625, 4697	\c_@@_tabularx_loaded_bool ...	46, 47, 3074
\l_@@_rows_seq	2654, 2656, 2657, 2659, 2661, 2664, 2666	\@@_test_hline_in_block:nnnnn	5097, 5099, 5504
\l_@@_rows_tl	4540, 4588, 4664, 4681, 4729, 4754	\@@_test_hline_in_stroken_block:nnnn ..	5101, 5526
\l_@@_rule_width_dim	269, 4885, 4983, 5033, 5051, 5171, 5219, 5253, 5402, 5415, 5422, 5439, 5460, 5474, 5478, 5491, 5498	\@@_test_if_cell_in_a_block:nn	5582, 5600, 5618, 5638
\l_@@_rules_color_tl ..	302, 620, 1603, 7155	\@@_test_if_cell_in_block:nnnnnnn ...	5644, 5646
\@@_set_CT@arc@:n	188, 197, 1603, 4878, 7155	\@@_test_if_math_mode:	277, 1568, 3007
\@@_set_CT@drsc@:n	198, 204, 4880	\@@_test_in_corner_h:	5102, 5123
\@@_set_final_coords:	3512, 3537	\@@_test_in_corner_v:	4914, 4935
\@@_set_final_coords_from_anchor:n ..	3528, 3617, 3648, 3729, 3738, 3804, 3852	\@@_test_vline_in_block:nnnnn	4909, 4911, 5515
\@@_set_initial_coords:	3507, 3526	\@@_test_vline_in_stroken_block:nnnn ..	4913, 5537
\@@_set_initial_coords_from_anchor:n ..	3517, 3610, 3645, 3728, 3735, 3798, 3846	\l_@@_the_array_box	1356, 1370, 1629, 1637, 2475, 2476, 2478, 2481
\@@_set_preamble:Nn	276, 862, 876, 2277, 6732	\c_@@_tikz_loaded_bool	51, 56, 1410, 3190, 6142, 7950
\@@_set_size:n	6725, 6742	\l_@@_tikz_rule_bool	5336, 5340, 5381, 5394, 5401, 5467, 5476
\l_@@_siunitx_loaded_bool ...	219, 223, 228	\l_@@_tikz_rule_tl	4882, 4967, 5054, 5055, 5155, 5256, 5257
\g_@@_size_seq	1331, 1336, 1386, 1387, 1388, 1389, 3103	\l_@@_tikz_seq	338, 6143, 6302, 6311
\l_@@_small_bool	845, 880, 886, 908, 943, 1234, 2123, 2152, 2910, 2956, 3176	\g_@@_tmpc_dim	7506, 7524, 7526, 7546
\@@_standard_cline	145, 1279	\l_@@_tmpc_dim	316, 1523, 1530, 3300, 3304, 4750, 4751, 4774, 4989, 5000, 5008, 5013, 5019, 5053, 5057, 5177, 5194, 5199, 5205, 5255, 5259, 6336, 6341, 6396, 6518, 6529, 6655, 6660, 6665, 6842, 6845, 6853
\@@_standard_cline:w	145, 146	\l_@@_tmpc_int	4653, 4654, 4655
\l_@@_standard_cline_bool ..	531, 633, 1278	\l_@@_tmpc_tl ...	4583, 4585, 4588, 4747, 4766, 6551, 6563, 6574, 6579, 6605, 6635, 6652
\c_@@_standard_tl	548, 549, 3879	\g_@@_tmpd_dim	7507, 7526, 7534
\l_@@_start_int	4862, 4904, 5092	\l_@@_tmpd_dim	317, 1525, 1531, 3302, 3305, 4771, 4774, 5001, 5008, 5187, 5194, 6338, 6341, 6374, 6385, 6389, 6392, 6396, 6527, 6530, 6844, 6845, 6863
\g_@@_static_num_of_col_int	335, 1730, 1780, 6212, 7696, 7711, 7987	\l_@@_tmpd_tl	4584, 4586, 4589, 6552, 6556, 6575, 6586, 6606, 6631, 6670
\l_@@_stop_loop_bool	3337, 3338, 3370, 3383, 3392, 3405, 3406, 3438, 3451, 3460	\g_@@_tmpe_dim	7508, 7527, 7546
\@@_stroke_block:nnn	6271, 6489	\g_@@_total_X_weight_int	303, 1227, 1616, 1619, 1638, 2224
\@@_stroke_borders_block:nnn ...	6283, 6593	\l_@@_total_width_bool ...	5403, 5468, 5471
\@@_stroke_borders_block_i: ...	6610, 6615	\@@_transform_preamble:	1751, 2626, 2676, 6746
\@@_stroke_borders_block_ii: ...	6618, 6622	\g_@@_types_of_matrix_seq	7632, 7633, 7638, 7642
\@@_stroke_horizontal:n ..	6633, 6635, 6668	\@@_underbrace_i:n	7434, 7472
\@@_stroke_vertical:n	6629, 6631, 6650	\@@_update_for_first_and_last_row: ..	982, 1047, 1340, 2930, 2975
\@@_sub_matrix:nnnnnnn	7052, 7075	\@@_use_arraybox_with_notes: ...	1677, 2543
\@@_sub_matrix_i:nnnn	7116, 7122	\@@_use_arraybox_with_notes_b: .	1674, 2527
\l_@@_submatrix_extra_height_dim	352, 6962, 7150	\@@_use_arraybox_with_notes_c:	1675, 1705, 2471, 2541, 2580
\l_@@_submatrix_hlines_clist	357, 6974, 6994, 7188, 7190	\l_@@_v_center_bool ..	6189, 6233, 6238, 6413
\l_@@_submatrix_left_xshift_dim	353, 6964, 7201, 7234	\@@_v_custom_line:n	5365, 5487
\l_@@_submatrix_name_str	7009, 7084, 7223, 7225, 7237, 7239, 7247, 7251	\c_@@_varwidth_loaded_bool ...	34, 35, 2064
\g_@@_submatrix_names_seq	326, 3211, 7006, 7010, 7920	\@@_vline:n	1915, 4889, 5075, 5494, 6560
\l_@@_submatrix_right_xshift_dim	354, 6966, 7210, 7244		
\g_@@_submatrix_seq ...	334, 1349, 3488, 7036		
\l_@@_submatrix_slim_bool	6972, 7092, 7845		
\l_@@_submatrix_vlines_clist	358, 6976, 6996, 7171, 7173		
\l_@@_suffix_tl	5779, 5790, 5800, 5803, 5852, 5860, 5861, 5879, 5887, 5888		
\l_@@_tabular_width_dim	268, 1131, 1133, 1819, 3086		

<code>\l_@@_vline_i:</code>	4896, 4899	<code>\l_@@_y_initial_dim</code>	313, 3510, 3608, 3618, 3656, 3657, 3661, 3663, 3670, 3680, 3681, 3813, 3818, 3861, 3866, 3906, 3923, 3932, 3969, 4027, 4040, 4042, 4054, 4057, 4399, 5036, 5219, 5220, 6911, 6935, 6949, 7125, 7136, 7137, 7149, 7165, 7182, 7226, 7235, 7245
<code>\l_@@_vline_ii:</code>	4924, 4932, 4960	<code>\l</code>	21, 2642, 2668, 6760, 6766, 6772, 6891, 7563, 7564, 7593, 7602, 7660, 7662, 7667, 7673, 7682, 7692, 7702, 7704, 7709, 7711, 7712, 7717, 7718, 7723, 7724, 7729, 7730, 7735, 7740, 7746, 7752, 7758, 7759, 7764, 7766, 7781, 7789, 7791, 7796, 7798, 7803, 7809, 7811, 7817, 7821, 7826, 7828, 7829, 7834, 7841, 7846, 7851, 7852, 7857, 7859, 7864, 7869, 7875, 7878, 7883, 7884, 7889, 7891, 7896, 7898, 7903, 7906, 7911, 7913, 7914, 7924, 7927, 7932, 7938, 7940, 7945, 7951, 7956, 7963, 7965, 7971, 7973, 7979, 7981, 7985, 7991, 7996, 7998, 8003, 8006, 8011, 8014, 8019, 8021, 8026, 8028, 8033, 8041, 8048, 8055, 8064, 8065, 8070, 8072, 8077, 8078, 8083, 8085, 8090, 8092, 8102, 8105, 8110, 8112, 8113, 8123, 8124, 8125, 8138, 8139, 8140, 8155, 8159, 8160, 8161, 8179, 8181, 8182, 8197, 8199, 8200, 8243, 8245, 8246, 8299, 8301, 8302, 8355, 8357, 8358, 8410, 8416, 8425
<code>\l_@@_vline_iii:</code>	4968, 4972	<code>\{</code>	294, 1846, 2143, 2168, 3014, 6790, 7206, 7904, 8006, 8245, 8357
<code>\l_@@_vline_iv:</code>	4965, 5027	<code>\}</code>	294, 1849, 2143, 2153, 3014, 6790, 7215, 7904, 8006, 8245, 8357
<code>\l_@@_vline_v:</code>	4969, 5043	<code>\sqcup</code>	7656, 7676, 7685, 7695, 7696, 7710, 7711, 7819, 7820, 7828, 7836, 7843, 7847, 7865, 7897, 7905, 7907, 7912, 7919, 7933, 7986, 7987, 7988, 7997, 8004, 8012, 8027, 8057, 8058, 8071, 8112
<code>\l_@@_vlines_block:nnn</code>	6245, 6546	<code>\l</code>	3016, 6789
<code>\l_@@_vlines_block_bool</code>	349, 6159, 6241, 6262		
<code>\l_@@_vlines_clist</code>	356, 663, 675, 680, 1229, 1765, 1771, 1802, 1814, 2256, 2263, 2833, 3189, 5073, 5074		
<code>\l_@@_vpos_col_str</code>	1937, 1940, 1942, 1947, 1969, 1985, 2061, 2214		
<code>\l_@@_vpos_of_block_tl</code>	346, 347, 5905, 5907, 6005, 6019, 6030, 6109, 6127, 6180, 6182		
<code>\l_@@_w:</code>	1756, 1841, 2344		
<code>\l_@@_weight_int</code>	2210, 2215, 2216, 2219, 2221, 2222, 2224, 2228		
<code>\l_@@_width_dim</code>	261, 840, 916, 1629, 1637, 3055, 3056, 3076, 3077		
<code>\g_@@_width_first_col_dim</code>	320, 1562, 1668, 2718, 2931, 2932		
<code>\g_@@_width_last_col_dim</code>	319, 1561, 1725, 2876, 2976, 2977		
<code>\l_@@_width_used_bool</code>	327, 917, 1614		
<code>\l_@@_x_final_dim</code>	314, 3514, 3563, 3572, 3573, 3576, 3579, 3580, 3731, 3747, 3755, 3759, 3763, 3765, 3770, 3772, 3802, 3811, 3819, 3859, 3867, 3909, 3924, 3933, 3967, 4022, 4032, 4401, 5034, 5232, 5235, 5237, 5239, 7091, 7107, 7108, 7114, 7210, 7227, 7244, 7413, 7420, 7421, 7427, 7430, 7446, 7463, 7478, 7493		
<code>\l_@@_x_initial_dim</code>	312, 3509, 3541, 3550, 3551, 3554, 3557, 3558, 3731, 3746, 3747, 3754, 3759, 3763, 3765, 3767, 3770, 3772, 3811, 3819, 3859, 3867, 3906, 3923, 3933, 3967, 4022, 4030, 4032, 4054, 4056, 4398, 5033, 5034, 5222, 5225, 5227, 5229, 7090, 7100, 7101, 7111, 7201, 7226, 7234, 7392, 7399, 7400, 7406, 7409, 7446, 7463, 7478, 7493		
<code>\l_@@_xdots_color_tl</code>	574, 588, 3597, 3636, 3717, 3718, 3786, 3834, 3888, 4262, 4337, 4360		
<code>\l_@@_xdots_down_tl</code>	613, 3895, 3917, 3952		
<code>\l_@@_xdots_inter_dim</code>	534, 536, 611, 3179, 3987, 3994, 4005, 4016, 4023, 4028, 4035, 4045, 5229, 5239		
<code>\l_@@_xdots_line_style_tl</code>	547, 549, 584, 3879, 3888		
<code>\l_@@_xdots_radius_dim</code>	544, 546, 607, 3178, 3618, 3619, 4055, 5474		
<code>\l_@@_xdots_shorten_end_dim</code>	538, 542, 594, 601, 3182, 3183, 3903, 4004, 4014, 4036, 4046		
<code>\l_@@_xdots_shorten_start_dim</code>	537, 541, 593, 598, 3180, 3181, 3904, 3993, 4014, 4036, 4046		
<code>\l_@@_xdots_up_tl</code>	614, 3894, 3916, 3942		
<code>\l_@@_y_final_dim</code>	315, 3515, 3615, 3619, 3657, 3661, 3663, 3688, 3698, 3699, 3813, 3816, 3861, 3864, 3909, 3924, 3932, 3969, 4027, 4042, 4402, 5038, 5220, 6913, 6935, 6949, 7128, 7143, 7144, 7149, 7166, 7183, 7227, 7235, 7245		
		A	
		<code>\A</code>	7004
		<code>\aboverulesep</code>	2518
		<code>\addtocounter</code>	471
		<code>\alpha</code>	405
		<code>\anchor</code>	3286, 3287
		<code>\array</code>	1542
		<code>\arraybackslash</code>	2016, 2239, 2378
		<code>\arraycolor</code>	1424
		<code>\arraycolsep</code>	693, 695, 697, 1130, 1237, 1361, 1362, 1704, 1708, 2476, 2844, 2850, 2860, 5227, 5237
		<code>\arrayrulecolor</code>	121
		<code>\arrayrulewidth</code>	153, 158, 178, 622, 973, 1155, 1158, 1164, 1195, 1699, 1710, 1768, 1774, 1860, 1907, 2259, 2266, 2535, 2574, 2730, 2732, 2738, 2748, 2750, 2756, 2779, 2781, 2787, 2805, 2808, 2814, 2842, 2848, 2860, 2863, 4750, 4751, 4753, 4769, 4771, 4985, 4999, 5000, 5003, 5016, 5022, 5075, 5173, 5189, 5202, 5208, 5277, 5295, 5480, 5692, 6493, 6548, 6571, 6588, 6595, 6935, 7150, 7154
		<code>\arraystretch</code>	1236, 3671, 3689, 5992, 6102, 6119, 7126, 7129
		<code>\AutoNiceMatrix</code>	6791
		<code>\AutoNiceMatrixWithDelims</code>	6738, 6783, 6795

B

`\baselineskip` 124, 130, 2044
`\begingroup` 2289
`\bfseries` 4499, 4502
`\bggroup` 1556
`\Block` 1290, 7947, 7958, 8012, 8043, 8092
`\BNiceMatrix` 6722
`\bNiceMatrix` 6719
`\Body` 1378
`\boldmath` 4499, 4502
bool commands:
`\bool_const:Nn` 34, 35, 37,
38, 40, 41, 43, 44, 46, 47, 51, 56, 62, 65, 68, 69
`\bool_do_until:Nn` 3338, 3406
`\bool_gset_false:N`
.... 1011, 1058, 1059, 1178, 1312, 1433,
1563, 1587, 1764, 1793, 2942, 2989, 3004,
3036, 3045, 5513, 5524, 5535, 5546, 6044, 6781
`\bool_gset_true:N` 1590, 1922,
2723, 2906, 2951, 2994, 3052, 4079, 4095,
4111, 4135, 4158, 4169, 4343, 4907, 5095, 6794
`\bool_if:NTF` 187,
442, 769, 778, 940, 943, 1039, 1151, 1179,
1230, 1234, 1239, 1300, 1301, 1330, 1335,
1350, 1407, 1410, 1435, 1438, 1440, 1460,
1553, 1555, 1569, 1579, 1614, 1647, 1723,
1728, 1749, 1754, 1782, 1794, 2023, 2086,
2123, 2152, 2404, 2514, 2709, 2726, 2744,
2762, 2775, 2801, 2838, 2872, 2877, 2910,
2928, 2956, 2973, 3074, 3095, 3097, 3099,
3161, 3176, 3190, 3660, 3662, 3805, 3853,
4077, 4093, 4109, 4133, 4156, 4493, 4635,
4658, 5226, 5236, 5340, 5344, 5471, 5476,
5591, 5609, 5627, 5675, 5685, 5707, 5991,
5995, 6039, 6101, 6105, 6118, 6122, 6233,
6241, 6251, 6327, 6354, 6398, 6425, 6462,
6802, 7390, 7411, 7605, 7615, 7647, 7845, 7915
`\bool_if:nTF` 228, 411, 438,
1117, 1657, 3493, 4374, 4713, 5064, 5068,
5266, 5270, 5347, 5701, 5952, 6261, 6412,
6452, 6914, 6924, 6926, 6939, 6944, 6953, 7950
`\bool_lazy_all:nTF` ... 1798, 1810, 2829,
4990, 5178, 5315, 5506, 5517, 5528, 5539, 5998
`\bool_lazy_and:nnTF`
.... 2473, 2765, 2843, 2849, 2857,
2911, 3649, 3915, 4175, 4724, 6511, 7175, 7192
`\bool_lazy_or:nnTF` 581, 1051, 1109, 1790,
2456, 2485, 2563, 2959, 3726, 3878, 3973,
4705, 4737, 4741, 4791, 4795, 4865, 5583,
5601, 5619, 5927, 5932, 7079, 7375, 7404, 7425
`\bool_lazy_or_p:nn` 2914
`\bool_not_p:n` 1801,
1803, 1813, 1815, 2767, 2832, 2834, 7405, 7426
`\bool_set:Nn` 3730
`\c_false_bool` .. 35, 38, 41, 44, 47, 56, 69,
2163, 2171, 2184, 2190, 2196, 4073, 4089, 4105
`\g_tmpa_bool`
.... 4907, 4915, 4942, 4950, 4955, 5095,
5103, 5130, 5138, 5143, 5513, 5524, 5535, 5546
`\g_tmpb_bool` 1764, 1794, 1922
`\l_tmpb_bool` ... 5588, 5602, 5620, 5642, 5655
`\c_true_bool`
.... 34, 37, 40, 43, 46, 51, 62, 65, 68, 2142

box commands:

`\box_clear_new:N` 1232, 1356
`\box_dp:N` 967, 987, 1024, 1044, 1099,
1260, 1269, 1345, 2383, 2425, 3689, 6074, 7129
`\box_gclear_new:N` 5981
`\box_grotate:Nn` 6041
`\box_ht:N` 968, 989, 995, 1007, 1030,
1042, 1091, 1262, 1264, 1267, 1343, 2011,
2035, 2037, 2043, 2379, 2424, 3671, 6065, 7126
`\box_ht_plus_dp:N` 2593, 2606, 7535
`\box_move_down:nn` 1099, 2041
`\box_move_up:nn`
.... 90, 92, 94, 1091, 2468, 2541, 2580
`\box_rotate:Nn` 1001
`\box_scale:Nnn` 7539
`\box_set_dp:Nn` 1023, 1043, 2425
`\box_set_ht:Nn` 1029, 1041, 2424
`\box_set_wd:Nn` 1017, 2475
`\box_use:N` 474, 1008, 1098, 2046, 7552
`\box_use_drop:N`
.... 1049, 1055, 1074, 2088, 2406,
2427, 2468, 2469, 2481, 2937, 6084, 6447, 6484
`\box_wd:N` ... 475, 1018, 1046, 1053, 1112,
1366, 1368, 1629, 1637, 2476, 2478, 2600,
2612, 2932, 2935, 2977, 2981, 6053, 6809, 7534
`\l_tmpa_box` 463, 474, 475, 1365, 1366,
1367, 1368, 1692, 2424, 2425, 2427, 2468,
2469, 2593, 2606, 7520, 7534, 7535, 7539, 7552
`\l_tmpb_box` 2586, 2600, 2601, 2612

C

`\c` 73, 1786
`\Cdots` 1282, 4084, 4087
`\cdots` 1211, 1274
`\cellcolor` 1220, 1418
`\centering` 1980, 6009
char commands:
`\char_set_catcode_space:n` 1742
`\chessboardcolors` 1426
`\cline` 181, 1279, 1280
clist commands:
`\clist_clear:N` 380, 4784
`\clist_if_empty:NnTF` 4914, 5102, 6279
`\clist_if_empty_p:N` 2833
`\clist_if_in:NnTF` 378, 1187, 1771,
2263, 5074, 5276, 6628, 6630, 6632, 6634, 6671
`\clist_map_inline:Nn`
.... 381, 4731, 4754, 4785, 5551, 7173, 7190
`\clist_map_inline:nn`
.... 3032, 4594, 4643, 5440, 6698
`\clist_new:N` 339, 355, 356, 357, 358, 556
`\clist_put_right:Nn` 390, 4802
`\clist_set:Nn` 675, 676, 680, 681
`\clist_set_eq:NN` 4783
`\l_tmpa_clist` 380, 390, 392, 4783, 4785
`\CodeAfter`
.... 935, 1294, 2645, 2651, 2905, 2950, 3210, 7905
`\CodeBefore` 1551
`\color` 125, 131, 193, 194, 201, 202, 215,
945, 3591, 3594, 3597, 3630, 3633, 3636,
3711, 3714, 3718, 3786, 3834, 4256, 4259,
4262, 4331, 4334, 4337, 4360, 4531, 7386, 7521
`\colorlet` ... 285, 286, 952, 959, 1226, 2920, 2964
`\columncolor` 1222, 1425, 3219, 4834

fp commands:		
\fp_eval:n	3928	1730, 2034, 2219, 2299, 2326, 2497, 2536,
\fp_min:nn	7531, 7533	2575, 2660, 2671, 2704, 2825, 2957, 3348,
\fp_set:Nn	7529	3355, 3359, 3361, 3416, 3423, 3427, 3429,
\fp_to_dim:n	3963	3593, 3632, 3713, 3748, 3750, 4258, 4333,
\fp_use:N	7539	4447, 4700, 4745, 4763, 4799, 4830, 4847,
\l_tmpa_fp	7529, 7539	4848, 4855, 4856, 4895, 4917, 4921, 4929,
\futurelet	140	5105, 5109, 5117, 5223, 5233, 5648, 5650,
		5652, 5654, 5987, 5989, 6046, 6058, 6235,
		6418, 6450, 6454, 6520, 6522, 6756, 6758,
		6760, 6764, 6766, 6768, 6770, 6772, 7159, 7161
G		
\globaldefs	1734, 6217	\int_compare_p:n
group commands:		3495, 3496, 3497, 3498, 4715, 4717, 6512, 6513
\group_insert_after:N	6803, 6804, 6806, 6807	\int_do_until:nNnn
		4655
H		\int_gadd:Nn
\halign	1271, 7457, 7489	2224, 2325
\hbox	1149, 1700, 2479,	\int_gincr:N
	2541, 2580, 2728, 2746, 2773, 2777, 2803, 2840	
hbox commands:		936, 965, 1578, 1881, 1998, 2091, 2116,
\hbox:n	90, 92, 95	2247, 2799, 2828, 2952, 3807, 3855, 5672, 5968
\hbox_gset:Nn	5983	\int_gset:Nn
\hbox_overlap_left:n	1092, 1100, 2707, 2933	1057, 1307, 1354, 2792
\hbox_overlap_right:n	474, 2874, 2979	\int_if_even:nTF
\hbox_set:Nn	463, 1089, 1365,	4603, 7279
	1367, 1692, 2039, 2238, 2586, 2601, 6326, 7520	\int_incr:N
\hbox_set:Nw	939, 1370, 2077, 2395, 2908, 2954	1891, 4683
\hbox_set_end:		\int_min:nn
	1037, 1613, 2085, 2403, 2927, 2972	3295,
\hbox_to_wd:nn	499, 524, 7462, 7492	3297, 3299, 3301, 3311, 3313, 3476, 3503, 3504
\Hdotsfor	1288, 7656, 7933	\int_mod:nn
\hdotsfor	1216	4671
\heavyrulewidth	2519	\int_step_inline:nn
\hfil	1842, 2345, 7459, 7491	1491, 1497, 3133, 3139, 3147,
\hfill	153, 178	3153, 3293, 4599, 4601, 7172, 7189, 7509, 7518
\Hline	1286, 5285	\int_step_inline:nnnn
\hline	133	5580, 5598, 5613, 5616
hook commands:		\int_use:N
\hook_gput_code:nnn	31, 99,	454
	115, 220, 226, 275, 409, 535, 539, 545, 558,	\c_zero_int
	591, 597, 600, 606, 610, 767, 776, 1303,	2124, 4516, 4706
	3254, 4061, 4191, 4268, 4352, 4385, 6613, 7044	
\hrule	137, 153, 178, 1195, 2519, 6948, 7312, 7335	io\ commands:
\hspace	2023	\iow_now:Nn
\hskip	136	76, 102, 1741, 1742, 1743, 1748
\Hspace	1287	\iow_shipout:Nn
\hspace	4170	5687, 5688, 5694
\hss	1842, 2345	\item
		2502, 2508
I		
\ialign	1138, 1247, 1270, 3202, 6196	
\iddots	1285, 4141, 4142, 4147, 4148	K
\iddots	85, 1214, 1277	
if commands:		\kern
\if_mode_math:	279, 4497	95
\IfBooleanTF	1604	keys commands:
\ifnum	135, 4459, 5280, 5306	\keys_define:nn
\ifstandalone	1575	30, 577,
\ignorespaces	2328, 4509	618, 626, 706, 747, 787, 794, 838, 868, 884,
int commands:		901, 914, 1443, 1927, 2209, 4161, 4405,
\int_case:nnTF	4115, 4121, 4139, 4145	4616, 4858, 4871, 5308, 5324, 5362, 5374,
\int_compare:nNnTF		5396, 5661, 5891, 6139, 6537, 6637, 6688,
	148, 149, 173, 937, 938, 949, 956, 992,	6730, 6875, 6880, 6960, 6981, 6990, 7357, 7575
	1002, 1141, 1192, 1194, 1327, 1333, 1338,	\keys_if_exist:nnTF
	1616, 1619, 1645, 1649, 1660, 1664, 1665,	1864
		\l_keys_key_str
		2210, 7563, 7760, 7765,
		7784, 7791, 7925, 7946, 8078, 8091, 8111,
		8124, 8139, 8160, 8180, 8198, 8244, 8300, 8356
		\keys_set:nn
		629, 631, 645, 857, 859, 867,
		1447, 1455, 1600, 1601, 1866, 1961, 2063,
		2110, 2218, 3038, 3047, 3058, 3078, 3087,
		3232, 3596, 3635, 3716, 3785, 3833, 4261,
		4336, 4359, 4438, 4632, 4963, 5151, 5339,
		5674, 6232, 6601, 6882, 6886, 7015, 7085, 7385
		\keys_set_known:nn
		4127, 4151, 5470, 5944, 6494, 6549, 6572, 6596
		\keys_set_known:nnN
		2217, 4894, 5083, 5314, 6745
		\l_keys_value_tl
		7827, 7967, 7975, 8411
		L
		\Large
		7521
		\Ldots
		1281, 4068, 4071
		\ldots
		1210, 1273
		\leaders
		153, 178

\left 1696, 2589, 2604, 6944, 7308, 7331
 legacy commands:
 \legacy_if:nTF 731
 \line 3207, 7904, 7907, 8134
 \linewidth 3056

M

\makebox 2088, 2406
 \mathinner 87
 mode commands:
 \mode_leave_vertical: 1567, 2377, 4489
 msg commands:
 \msg_error:nn 11
 \msg_error:nnn 12
 \msg_error:nnnn 14, 6214, 6221, 6225
 \msg_fatal:nn 15
 \msg_fatal:nnn 16
 \msg_new:nnn 17, 21
 \msg_new:nnnn 22, 7561
 \msg_redirect_name:nnn 27
 \multicolumn . 1302, 1304, 4172, 4181, 4187, 4209
 \multispan 149, 150, 174, 175, 2288
 \myfiledate 6
 \myfileversion 7

N

\newcolumnntype 3074
 \newcounter 395, 399
 \NewDocumentCommand
 413, 436, 866, 1453, 3231, 4065,
 4081, 4097, 4113, 4137, 4195, 4272, 4356,
 4433, 4544, 4553, 4562, 4571, 4592, 4597,
 4610, 4625, 4696, 4805, 4816, 4828, 6738,
 6791, 7022, 7031, 7048, 7057, 7347, 7352, 7504
 \NewDocumentEnvironment .. 1550, 2619, 2635,
 2992, 3002, 3034, 3043, 3053, 3072, 3083, 5670
 \NewExpandableDocumentCommand . 244, 5432, 5914
 \newlist 417, 428
 \NiceArray 3060, 3080, 3089
 \NiceArrayWithDelims 2997, 3008
 nicematrix commands:
 \g_nicematrix_code_after_tl
 296, 740, 2653,
 3212, 3214, 6243, 6253, 6269, 6281, 6894, 6901
 \g_nicematrix_code_before_tl 1319, 3216,
 3223, 4441, 4449, 4809, 4820, 4832, 6291, 6304
 \NiceMatrixLastEnv 244
 \NiceMatrixOptions 866, 8199, 8416
 \NiceMatrixoptions 7972
 \noalign 124, 130, 135,
 158, 1174, 1253, 5280, 5412, 5436, 7461, 7495
 \nobreak 431
 \nointerlineskip 7461, 7495
 \normalbaselines 1233
 \normalfont 7521
 \NotEmpty 1296
 \null 2324
 \nulldelimiterspace 2600, 2612, 6930, 7152
 \nullfont 6941, 6947, 7305, 7311, 7328, 7334

O

\omit 148, 2706, 2722, 2798, 2824, 6890, 6891
 \OnlyMainNiceMatrix 1292, 4839
 \OverBrace 3205, 7381, 8112

P

\par 2496, 2504
 \path 6700
 peek commands:
 \peek_meaning:NTF 461, 1204, 2623, 3068, 5285
 \peek_meaning_remove_ignore_spaces:NTF 181
 \peek_remove_spaces:n 2621,
 4807, 4818, 5283, 5916, 7024, 7050, 7349, 7354
 \pgfdeclareshape 3279
 \pgfextracty 6421
 \pgfgetlastxy 517
 \pgfpathcircle 4053
 \pgfpathlineto 5013, 5019,
 5199, 5205, 6660, 6680, 6845, 7166, 7183, 7217
 \pgfpathmoveto 5012, 5018,
 5198, 5204, 6659, 6679, 6840, 7165, 7182, 7208
 \pgfpathrectanglecorners 4773, 5006, 5192, 6528
 \pgfpointhead 515
 \pgfpointhead 218, 247, 3519,
 3530, 3547, 3569, 3677, 3695, 5733, 5741,
 6364, 6382, 6402, 6404, 6422, 6459, 6922,
 7099, 7106, 7135, 7142, 7253, 7257, 7398, 7419
 \pgfpointhead 516, 1398, 1404, 1472, 1486, 1487
 \pgfpointhead 3922
 \pgfpointhead 2713, 2886, 3287
 \pgfpointhead 515
 \pgfpointhead 4397, 4400
 \pgfrememberpicturerepositiononpagetrue ...
 970, 1065, 1162,
 1509, 2712, 2736, 2754, 2785, 2812, 2854,
 2883, 3265, 3292, 3876, 4396, 4975, 5030,
 5046, 5163, 5216, 5248, 5776, 5786, 5797,
 6329, 6496, 6624, 6835, 6908, 7087, 7388, 7541
 \pgfscope 3919, 6852
 \pgfset 484,
 509, 1066, 6436, 6473, 6851, 6929, 7089, 7432
 \pgfsetbaseline 1064
 \pgfsetcornersarced 4772, 6504
 \pgfsetlinewidth .. 5022, 5208, 6531, 6627, 7154
 \pgfsetrectcap 5023, 5209
 \pgfsetroundcap 6848
 \pgfsyspdfmark 79, 80
 \pgftransformrotate 3926
 \pgftransformshift
 490, 515, 3303, 3920, 6435, 6457,
 6853, 6863, 6931, 7231, 7241, 7443, 7475, 7543
 \pgfusepath 3946, 3956, 4534, 5009, 6532
 \pgfusepathqfill 4059, 5195
 \pgfusepathqstroke
 5024, 5210, 6661, 6681, 6849, 7167, 7184, 7218
 \phantom 4078, 4094, 4110, 4134, 4157
 \pNiceMatrix 6710
 prg commands:
 \prg_do_nothing: 83, 229, 615, 1253, 3210, 6746
 \prg_new_conditional:Nnn 4703, 4711
 \prg_replicate:nn 2794,
 2795, 4209, 5014, 5200, 6759, 6762, 6765, 6771
 \prg_return_false: 4708, 4720
 \prg_return_true: 4709, 4719
 \ProcessKeysOptions 7586
 \ProvideDocumentCommand 85
 \ProvidesExplPackage 4

Q	
\quad	432
quark commands:	
\q_stop	145, 146, 162, 163, 166, 167, 169, 170, 171, 371, 384, 1436, 1458, 1779, 1852, 1922, 2156, 2178, 2283, 2295, 2346, 2645, 2651, 4344, 4348, 4350, 4364, 4365, 4582, 4587, 4647, 4735, 4736, 4758, 4759, 4789, 4790, 5450, 5451, 5870, 5877, 5919, 5920, 5924, 6510, 6519, 6550, 6553, 6573, 6576, 6604, 6607, 6725, 6742, 7264, 7272
R	
\raggedleft	1982, 6010, 6867
\raggedright	1981, 2023, 6011
\rectanglecolor	1419, 3218
\refstepcounter	472
regex commands:	
\regex_const:Nn	73
\regex_match:nnTF	7004
\regex_replace_all:NnN	1784
\renewcommand	233
\RenewDocumentEnvironment	6709, 6712, 6715, 6718, 6721
\RequirePackage	1, 3, 9, 10
\right	1713, 2596, 2608, 6953, 7316, 7339
\rotate	1291
\roundedrectanglecolor	1420, 6293
\rowcolor	1221, 1421
\rowcolors	1422
\rowlistcolors	1423
\RowStyle	1297, 8181
S	
\savedanchor	3281
\savenotes	1555
scan commands:	
\scan_stop:	3213
\scriptstyle	943, 2910, 2956, 3907, 3908, 3942, 3952
seq commands:	
\seq_clear:N	473, 1777
\seq_clear_new:N	2654, 2681, 4628, 5550
\seq_count:N	2661, 2687, 4519, 4673
\seq_gclear:N	1305, 1306, 1348, 1349, 1351, 1581, 1582, 1583, 1584, 2524, 3211
\seq_gclear_new:N	1352, 1353, 1432
\seq_gput_left:Nn	736, 2301, 2303, 6264
\seq_gput_right:Nn	455, 1857, 2304, 3473, 4518, 6079, 6091, 6276, 6822, 7010, 7036
\seq_gset_from_clist:Nn	3103, 3117, 3125, 3127, 7633
\seq_gset_map_x:NNn	3235, 7638
\seq_if_empty:NTF	2483, 3113, 3121, 5566, 6302
\seq_if_empty_p:N	2486, 4725
\seq_if_in:NnTF	734, 4765, 4939, 4945, 4952, 5127, 5133, 5140, 5736, 5744, 6362, 6380, 7006, 7642
\seq_item:Nn	1331, 1336, 1386, 1387, 1388, 1389, 4692, 4694
\seq_map_break:	447
\seq_map_indexed_inline:Nn	444, 4514, 4529
\seq_map_inline:Nn	1298, 1510, 2502, 2508, 2666, 2691, 3488, 4662, 4908, 4910, 4912, 5096, 5098, 5100, 5643, 6197, 7157
\seq_mapthread_function:NNN	5865
\seq_new:N	262, 284, 301, 321, 322, 323, 324, 325, 326, 328, 329, 334, 338, 396, 398, 7632
\seq_pop_left:NN	2664, 2689
\seq_pop_right:NN	2657
\seq_put_left:Nn	5427, 5445
\seq_put_right:Nn	453, 458, 2659, 5630, 6143
\seq_set_eq:NN	4637
\seq_set_filter:NNn	4638, 4660
\seq_set_from_clist:Nn	5570
\seq_set_split:Nnn	29, 2656, 2682, 4629
\seq_use:Nn	6311
\seq_use:Nnnn	467, 3118, 3126, 3128, 5571, 7920, 8421
\l_tmpa_seq	4638, 4660
\l_tmpb_seq	4637, 4638, 4660, 4662
\setcounter	402
\setlist	418, 429, 770, 779
\ShowCellNames	1428, 3206
siunitx commands:	
\siunitx_cell_begin:w	1986, 2104, 2111
\siunitx_cell_end:	1987, 2114
skip commands:	
\skip_gadd:Nn	2770
\skip_gset:Nn	2761, 2826
\skip_gset_eq:NN	2768, 2769
\skip_horizontal:N	154, 179, 1371, 1372, 1611, 1612, 1667, 1668, 1703, 1704, 1707, 1708, 1725, 1726, 1768, 1774, 1860, 2259, 2266, 2613, 2614, 2616, 2617, 2717, 2718, 2730, 2732, 2748, 2750, 2772, 2779, 2781, 2800, 2805, 2808, 2827, 2837, 2842, 2844, 2848, 2850, 2876, 2938, 2939, 2940, 2943, 2978, 2983, 2984, 2985
\skip_horizontal:n	475, 1112, 1903, 5491
\skip_vertical:N	158, 1155, 1158, 2493, 2518
\skip_vertical:n	1007, 1699, 1710, 5293, 5415, 5439, 7461, 7495
\g_tmpa_skip	2761, 2768, 2769, 2770, 2772, 2800, 2826, 2827
\c_zero_skip	1258
\smash	7340, 7341
\space	293, 294, 7946
\SplitArgument	7032, 7058
\stepcounter	452
str commands:	
\c_backslash_str	293
\str_case:nn	1978, 6007, 6428, 6440, 6465, 6477, 7202, 7211
\str_case:nnTF	1672, 1827, 1990, 2332, 2450, 5553, 7276, 7289
\str_clear_new:N	5311, 5312, 5313, 7084
\str_const:Nn	5373, 7627, 7629
\str_count:N	5355
\str_gclear:N	3227
\str_gset:Nn	1565, 2996, 3006, 3037, 3046, 3057, 3075, 3085, 6782
\str_if_empty:NTF	974, 1077, 1165, 1564, 2714, 2739, 2757, 2788, 2815, 2865, 2887, 2995,

3005, 3145, 3307, 3319, 5353, 5370, 5371, 5857, 5884, 5939, 6342, 6347, 7223, 7237, 7247	
\str_if_empty_p:N	5317, 5318, 5319
\str_if_eq:nnTF	24, 110, 292, 1135, 1855, 1896, 1922, 1952, 1969, 1972, 1985, 1986, 1987, 2053, 2096, 2126, 2158, 2180, 2203, 2253, 2413, 2640, 2642, 2697, 4424, 4692, 6500, 6899, 7038, 7039, 7040, 7041, 7379, 7433
\str_if_eq_p:nn	583, 1791, 1792, 4739, 4743, 4793, 4797, 4867, 5929, 5934
\str_if_in:NnTF	2438, 2550
\str_if_in:nnTF	5449
\str_new:N	256, 287, 343, 564
\str_range:Nnn	2442, 2554
\str_set:Nn	257, 344, 733, 1877, 1929, 1931, 1933, 1935, 1937, 1940, 1942, 1947, 1960, 1973, 1975, 2061, 2062, 2079, 2213, 2354, 2397, 5327, 5329, 5331, 5893, 5895, 5897, 5899, 5901, 5903, 5905, 5907, 5940, 5943, 5995, 6106, 6123, 6165, 6167, 6169, 6171, 6174, 6177, 6180, 6182, 6451, 6455, 7009
\str_set_eq:NN	737, 5941
\l_tmpa_str	733, 734, 736, 737
\strut	2502, 2508
\strutbox	3671, 3689, 7126, 7129
\SubMatrix	1427, 3203, 7027, 7082, 7828, 7836, 7843, 7847, 7912
sys commands:	
\sys_if_engine_xetex_p:	1109
\sys_if_output_dvi_p:	1109
\c_sys_jobname_str	24

T

\tabcolsep	1129, 1703, 1707
\tabskip	1258
\tabularnote	413, 436, 461, 8004, 8027
\tabularnotes	2507
T _E X and L ^A T _E X 2 _ε commands:	
\@BTnormal	1231
\@addamp	1536, 2290
\@addamp@LaTeX	1536
\@addtopreamble	2297
\@array	1543, 1547
\@array@array	1543
\@arraycr	1540
\@arraycr@array	1540
\@arstrut	2322
\@arstrutbox	967, 968, 1007, 1260, 1262, 1264, 1267, 1269, 2011, 2022, 2037, 2043, 2379, 2383
\@classx	1538
\@classx@array	1538
\@currentenv	6899
\@depth	6950, 7313, 7336
\@empty	2297
\@finalstrut	2022
\@firststampfalse	2290
\@gobblethree	80
\@halignto	1132, 1133
\@height	138, 153, 178, 6948, 7312, 7335
\@ifclassloaded	61, 64, 7607, 7617
\@ifnextchar	1310, 1547
\@ifpackageloaded	33, 36, 39, 42, 45, 49, 101, 117, 222, 7610, 7620

\@mainaux	76, 102, 1741, 1742, 1743, 1748, 5687, 5688, 5694
\@mkpream	1545, 2296
\@mkpream@array	1545
\@preamble	2323
\@preamerr	2290
\@sharp	2321
\@tabarray	1134, 1547
\@tabular	1544
\@tabular@array	1544
\@tempswafalse	1761, 2281, 2293
\@tempswatrue	1760, 2280, 2292
\@temptokena	235, 236, 1759, 1779, 2279, 2283, 2291, 2295
\@whiles	1761, 2281, 2293
\@width	138, 6951, 7314, 7337
\@xargarraycr	1541
\@xargarraycr@array	1541
\@xarraycr	1539
\@xarraycr@array	1539
\@xhline	141
\array@array	1542
\bBigg@	1365, 1367
\c@MaxMatrixCols	3025, 7687
\c@tabularnote	2497, 2525
\col@sep	1129, 1130, 1667, 1726, 2717, 2770, 2837, 2943, 2978, 3558, 3580
\CT@arc	121, 122
\CT@arc@	120, 125, 139, 152, 177, 193, 194, 1571, 2519, 3229, 5021, 5039, 5207, 5241, 6501, 6626, 6847, 7156
\CT@dr	127, 128
\CT@drsc@	131, 201, 202, 4993, 4994, 4998, 5181, 5182, 5186
\CT@everycr	1251
\CT@row@color	1253
\endarray@array	1546
\if@firststamp	2290
\if@tempswa	1761, 2281, 2293
\insert@column	1537
\insert@column@array	1537
\NC@	1203, 3067
\NC@do	3066
\NC@find	237
\NC@find@V	1758
\NC@list	1761, 2281, 2293, 3066
\NC@rewrite@S	233
\new@ifnextchar	1310
\newcol@	1205, 1206, 3069, 3070
\nicematrix@redefine@check@rerun	102, 105
\pgf@relevantforpicturesizefalse	1393, 1508, 3266, 3877, 4050, 4528, 4642, 4976, 5031, 5047, 5164, 5217, 5249, 5777, 5787, 5798, 6330, 6497, 6625, 6834, 6909, 7088, 7389, 7542
\pgfsys@getposition	1391, 1396, 1402, 1470, 1478, 1481
\pgfsys@markposition	1094, 1102, 1156, 1243, 1390, 2710, 2731, 2749, 2780, 2806, 2845, 2879
\pgfutil@check@rerun	107, 108
\reserved@a	140
\rvtx@ifformat@geq	67

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	Commands for customized rules	11
6	The color of the rows and columns	13
6.1	Use of <code>colortbl</code>	13
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	14
6.3	Color tools with the syntax of <code>colortbl</code>	18
7	The command <code>\RowStyle</code>	18
8	The width of the columns	19
8.1	Basic tools	19
8.2	The columns <code>V</code> of <code>varwidth</code>	20
8.3	The columns <code>X</code>	21
9	The exterior rows and columns	21
10	The continuous dotted lines	23
10.1	The option <code>nullify-dots</code>	24
10.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	25
10.3	How to generate the continuous dotted lines transparently	26
10.4	The labels of the dotted lines	26
10.5	Customisation of the dotted lines	27
10.6	The dotted lines and the rules	28
11	The <code>\CodeAfter</code>	28
11.1	The command <code>\line</code> in the <code>\CodeAfter</code>	29
11.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	29
11.3	The commands <code>\OverBrace</code> and <code>\UnderBrace</code> in the <code>\CodeAfter</code>	31

12	The notes in the tabulars	32
12.1	The footnotes	32
12.2	The notes of tabular	32
12.3	Customisation of the tabular notes	34
12.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	36
13	Other features	36
14	Autres fonctionnalités	36
14.1	Command <code>\ShowCellNames</code>	36
14.2	Use of the column type <code>S</code> of <code>siunitx</code>	36
14.3	Default column type in <code>{NiceMatrix}</code>	37
14.4	The command <code>\rotate</code>	37
14.5	The option <code>small</code>	38
14.6	The counters <code>iRow</code> and <code>jCol</code>	38
14.7	The key <code>light-syntax</code>	39
14.8	Color of the delimiters	39
14.9	The environment <code>{NiceArrayWithDelims}</code>	39
14.10	The command <code>\OnlyMainNiceMatrix</code>	40
15	Use of Tikz with <code>nicematrix</code>	40
15.1	The nodes corresponding to the contents of the cells	40
15.1.1	The columns <code>V</code> of <code>varwidth</code>	41
15.2	The medium nodes and the large nodes	41
15.3	The nodes which indicate the position of the rules	43
15.4	The nodes corresponding to the command <code>\SubMatrix</code>	44
16	API for the developpers	44
17	Technical remarks	45
17.1	Diagonal lines	45
17.2	The empty cells	46
17.3	The option <code>exterior-arraycolsep</code>	46
17.4	Incompatibilities	47
18	Examples	47
18.1	Utilisation of the key <code>'tikz'</code> of the command <code>\Block</code>	47
18.2	Notes in the tabulars	48
18.3	Dotted lines	49
18.4	Dotted lines which are no longer dotted	50
18.5	Dashed rules	51
18.6	Stacks of matrices	51
18.7	How to highlight cells of a matrix	55
18.8	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	57
19	Implementation	58
20	History	243
	Index	251