

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

August 10, 2021

Abstract

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

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

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

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

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

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

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

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 6.0 of `nicematrix`, at the date of 2021/08/10.

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

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

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

1 The environments of this package

The package `nicematrix` defines the following new environments.

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

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

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

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

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

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

2 The vertical space between the rows

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

```


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


```

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

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

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

```

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


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


```

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

3 The vertical position of the arrays

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

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

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

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

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

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

1. an item

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

However, it's also possible to use the tools of `booktabs`⁴: `\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}

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

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

⁴The 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.⁵

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

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

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

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

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

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

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

One may wish to raise the size of the “A” placed in the block of the previous example. Since this element is composed in math mode, it's not possible to use directly a command like `\large`, `\Large` and `\LARGE`. That's why the command `\Block` provides an option between angle brackets to specify some TeX code which will be inserted before the beginning of the math mode.⁶

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

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

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

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

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

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

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

⁵The spaces after a command `\Block` are deleted.

⁶This argument between angular brackets may also be used to insert a command of font such as `\bfseries` when the command `\\` is used in the content of the block.

- the key `fill` takes in as value a color and fills the block with that color;
- the key `draw` takes in as value a color and strokes the frame of the block with that color (the default value of that key is the current color of the rules of the array);
- the key `color` takes in as value a color and apply that color the content of the block but draws also the frame of the block with that color;
- the key `line-width` is the width (thickness) of the frame (this key should be used only when the key `draw` or the key `hvlines` is in force);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt⁷);
- the key `borders` provides the ability to draw only some borders of the blocks; the value of that key is a (comma-separated) list of elements covered by `left`, `right`, `top` and `bottom`;
- the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`);
- the keys `hvlines` draws all the vertical and horizontal rules in the block;
- **New 5.19** when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz⁸ 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. 42.

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

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

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

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

4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

- The natural width of the contents of these blocks is taken into account for the width of the current column.

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

⁷This value is the initial value of the *rounded corners* of Tikz.

⁸Tikz should be loaded (by default, `nicematrix` only loads PGF) and, if it's not, an error will be raised.

- 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`.⁹
- It's possible to draw one or several borders of the cell with the key `borders`.

<code>\begin{NiceTabular}{cc}</code>	
<code>\toprule</code>	
<code>Writer & \Block[l]{year of birth} \\\</code>	Writer year of birth
<code>\midrule</code>	
<code>Hugo & 1802 \\\</code>	Hugo 1802
<code>Balzac & 1799 \\\</code>	Balzac 1799
<code>\bottomrule</code>	
<code>\end{NiceTabular}</code>	

We recall that if the first mandatory argument of `\Block` is left blank, the block is mono-cell.¹⁰

⁹If 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).

¹⁰One may consider that the default value of the first mandatory argument of `\Block` is 1-1.

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

New 5.17 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.

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`.

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

However, in this case, it is probably more clever to add a command `\OnlyMainNiceMatrix` (cf. p. 40):

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

5.1.2 The command `\cline`

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

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

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

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

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

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

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

5.2 The thickness and the color of the rules

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

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

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

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

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

If one wishes to define new specifiers for columns in order to draw vertical rules (for example with a specific color or thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 40.

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`¹¹ presented p. 4;
 - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `\Vdots`, etc. (cf. p. 20).
- 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.

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

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

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

5.3.2 The 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

New 5.17 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.

¹¹And also the command `\multicolumn` but it’s recommended to use instead `\Block` in the environments of `nicematrix`.

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.¹²

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

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.

				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		
		B		A	
			A		

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

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

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

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

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

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

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

¹²For 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.

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.¹³

```
\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}
```

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

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

5.5 Dotted rules

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

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

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

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

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

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

It's possible to change in `nicematrix` the letter used to specify a vertical dotted line with the option `letter-for-dotted-lines` available in `\NiceMatrixOptions`. Thus released, the letter “:” can be used otherwise (for example by the package `arydshln`¹⁴).

Remark: In the package `array` (on which the package `nicematrix` relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule¹⁵. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

6 The color of the rows and columns

6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

¹³The author of this document considers that type of construction as graphically poor.

¹⁴However, one should remark that the package `arydshln` is not fully compatible with `nicematrix`.

¹⁵In fact, with `array`, this is true only for `\hline` and “|” but not for `\cline`: cf p. 8

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

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

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

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

The package `nicematrix` provides some tools (independent of `colortbl`) to draw the colored panels first, and, then, the content of the cells and the rules. This strategy is more conform to the “painting model” of the formats PostScript and PDF and is more suitable for the PDF viewers. However, it requires several compilations.¹⁶

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

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

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

New commands are available in that `\CodeBefore`: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors`, `\rowlistcolors`, `\chessboardcolors` and `arraycolor`.¹⁷

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.

¹⁶If you use Overleaf, Overleaf will do automatically the right number of compilations.

¹⁷Remark 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. 37.

```

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

```

a	b	c
e	f	g
h	i	j

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

```

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

```

a	b	c
e	f	g
h	i	j

- The command `\arraycolor` takes in as mandatory argument a color and color the whole tabular with that color (excepted the potential exterior rows and columns: cf. p. 18). 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. 32).

- 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).

```

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

```

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

- The command `\columncolor` takes its name from the command `\columncolor` of `colortbl`. Its syntax is similar to the syntax of `\rowcolor`.
- The command `\rowcolors` (with a *s*) takes its name from the command `\rowcolors` of `xcolor`¹⁸. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows of the tabular with the tow colors (provided in second and third argument), beginning with the row whose number is given in first (mandatory) argument.

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

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

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

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

¹⁸The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

¹⁹Otherwise, the color of a given row relies only upon the parity of its absolute number.

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

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

Peter
James
Abigail
Elisabeth
Claudius
Jane
Alexandra

We recall that all the color commands we have described don't color the cells which are in the "corners". In the following example, we use the key `corners=NE` 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{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.²⁰ There are three commands available (they are inspired by `colortbl` but are *independent* of `colortbl`):

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

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

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

7 The command `\RowStyle`

New 5.18 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 available keys are `cell-space-top-limit`, `cell-space-bottom-limit` and `cell-space-limits` with the same meaning that the corresponding global keys (cf. p. 2).

```
\begin{NiceTabular}{cccc}[hlines,colortbl-like]
\RowStyle[cell-space-limits=3pt]{\rotate}
first & second & third & fourth \\
\rowcolor{blue!50}\RowStyle{\color{white}\sffamily}
1 & 2 & 3 & 4 \\
\end{NiceTabular}
```

first	second	third	fourth
1	2	3	4

The command `\rotate` is described p. 33.

²⁰Up to now, this key is *not* available in `\NiceMatrixOptions`.

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. 18) 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.²¹

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

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

Without surprise, it's possible to fix the minimal width of the columns of all the arrays of a current scope with the command `\NiceMatrixOptions`.

```
\NiceMatrixOptions{columns-width=10mm}
$\begin{pNiceMatrix}
a & b \\ c & d
\end{pNiceMatrix}
=
\begin{pNiceMatrix}
1 & 1245 \\ 345 & 2
\end{pNiceMatrix}$
```

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

But it's also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment `{NiceMatrixBlock}` with the option `auto-columns-width`²². The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

²¹The 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).

²²At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

```

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

```

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

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

8.2 The columns X

New 6.0

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

The required width of the tabular may be specified with the key `width`. The initial value of this parameter is `\linewidth`.

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.

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

- It's possible to give a weight for the column by providing an 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).
- It's possible to specify an horizontal alignment with one of the letters `l`, `c` and `r` (which inserts 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”.

```

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

```

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

The dotted lines have been drawn with the tools presented p. 20.

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.²³
- 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. 34) is used, `nicematrix` has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (but not the number of columns).
 - In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

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

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

```

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

```

²³The users wishing exteriors columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 26).

```

& 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}$

```

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

Remarks

- As shown in the previous example, the horizontal and vertical rules don't extend in the exterior rows and columns.
However, if one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 40.
- A specification of color present in `code-for-first-row` also applies to a dotted line drawn in that exterior “first row” (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 17) doesn't apply to the “first column” and “last column”.
- For technical reasons, it's not possible to use the option of the command `\\` after the “first row” or before the “last row”. The placement of the delimiters would be wrong. If you are looking for a workaround, consider the command `\SubMatrix` in the `\CodeAfter` described p. 26.

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`.²⁴

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells²⁵ 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`.²⁶

```

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

```

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

²⁴The 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`.

²⁵The precise definition of a “non-empty cell” is given below (cf. p. 41).

²⁶It'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. 24.

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

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

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

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

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

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

In the first column of this example, 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 `\` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.²⁷

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}
```

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

10.1 The option `nullify-dots`

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

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

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

²⁷In `nicematrix`, one should use `\hspace*` and not `\hspace` for such an usage because `nicematrix` loads `array`. One may also remark that it's possible to fix the width of a column by using the environment `{NiceArray}` (or one of its variants) with a column of type `w` or `W`: see p. 17

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}
```

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}
```

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}
```

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}
```

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

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`²⁸ 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`:

²⁸We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

```

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

```

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

10.3 How to generate the continuous dotted lines transparently

Imagine you have a document with a great number of mathematical matrices with ellipsis. You may wish to use the dotted lines of `nicematrix` without having to modify the code of each matrix. It's possible with the keys `renew-dots` and `renew-matrix`.²⁹

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`²⁴ and `\hdotsfor` are redefined within the environments provided by `nicematrix` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` (“automatic dots” of `amsmath`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

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

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

```

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

```

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

²⁹The options `renew-dots`, `renew-matrix` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage`. There is also a key `transparent` which is an alias for the conjunction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.

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. 25) 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 \\\[8mm]
& \Ddots^{n \text{ times}} & & \\
0 & & & 1
\end{bNiceMatrix}$

```

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

10.5 Customisation of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and by the command `\line` in the `\CodeAfter` which is described p. 25) may be customized by three options (specified between square brackets after the command):

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

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

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

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

The option `xdots/color`

The option `xdots/color` fixes the color of the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 18.

The option `xdots/shorten`

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

The option `xdots/line-style`

It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).³⁰

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

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

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

³⁰The first reason of this behaviour is that the PDF format includes a description for dashed lines. The lines specified with this descriptor are displayed very efficiently by the PDF readers. It’s easy, starting from these dashed lines, to create a line composed by square dots whereas a line of rounded dots needs a specification of each dot in the PDF file.

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

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

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

10.6 The dotted lines and the rules

The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier `|` in the preamble, by the command `\Hline` and by the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` are not drawn within the blocks).³¹

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

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

11 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.³²

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

The experienced users may, for instance, use the PGF/Tikz nodes created by `nicematrix` in the `\CodeAfter`. These nodes are described further beginning on p. 35.

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

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

The command `\line` draws directly dotted lines between nodes. It takes in two arguments for the two cells to link, both of the form *i-j* where *i* is the number of the row and *j* is the number of the column. The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 24).

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

³¹On the other side, the command `\line` in the `\CodeAfter` (cf. p. 25) does *not* create block.

³²There is also a key `code-before` described p. 12.

```

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

```

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

It can also be used to draw a diagonal line not parallel to the other diagonal lines (by default, the dotted lines drawn by `\Ddots` are “parallelized”: cf. p. 40).

```

\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}

```

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

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

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

- the first argument is the left delimiter, which may be any extensible delimiter provided by LaTeX : `(`, `[`, `\{`, `\langle`, `\lgroup`, `\lfloor`, etc. but also the null delimiter `.`;
- 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.³³

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.

```

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

```

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

³³There 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}`).

New 5.18 In fact, the command `\SubMatrix` also takes in two optional arguments specified by the traditional symbols `^` and `_` for material in superscript and subscript.

```

 $\begin{bNiceMatrix}[right-margin=1em]
1 & 1 & 1 \\
1 & a & b \\
1 & c & d
\end{bNiceMatrix}$ 

```

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

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.

```

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

```

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

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

```

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

```

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

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. 38.

It's also possible to specify some delimiters³⁴ 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.

```

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

```

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

12 The notes in the tabulars

12.1 The footnotes

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

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

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

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

12.2 The notes of tabular

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

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

```

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

```

³⁴Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

```

& Jacques & 5 juin 1962 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.}
& Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}

```

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

^a Achard is an old family of the Poitou.

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

- If you have several successive commands `\tabularnote{...}` with no space at all between them, the labels of the corresponding notes are composed together, separated by commas (this is similar to the option `multiple` of `footmisc` for the footnotes).
- If a command `\tabularnote{...}` is exactly at the end of a cell (with no space at all after), the label of the note is composed in an overlapping position (towards the right). This structure may provide a better alignment of the cells of a given column.
- If the key `notes/para` is used, the notes are composed at the end of the array in a single paragraph (as with the key `para` of `threeparttable`).
- There is a key `tabularnote` which provides a way to insert some text in the zone of the notes before the numbered tabular notes.
- If the package `booktabs` has been loaded (before or after `nicematrix`), the key `notes/bottomrule` draws a `\bottomrule` of `booktabs` after the notes.
- The command `\tabularnote` may be used *before* the environment of `nicematrix`. Thus, it's possible to use it on the title inserted by `\caption` in an environment `{table}` of LaTeX.
- It's possible to create a reference to a tabular note created by `\tabularnote` (with the usual command `\label` used after the `\tabularnote`).

For an illustration of some of those remarks, see table 1, p. 30. This table has been composed with the following code.

```

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

```

```

Touchet & Marie & 89 \\
Wallis & John & 87 \\
\bottomrule
\end{NiceTabular}
\end{table}

```

Table 1: Use of `\tabularnote`^a

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

Some text before the notes.

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

^b Considered as the first nurse of history.

^c Nicknamed “the Lady with the Lamp”.

^d The label of the note is overlapping.

12.3 Customisation of the tabular notes

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

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

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

```

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

```

We detail these keys.

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

Initial value: `false`

That key is also available within a given environment.

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

Initial value: `false`

That key is also available within a given environment.

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

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

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

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

Initial value: `#1`

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

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

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

Initial value: `#1`

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

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

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

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

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

This initial value contains the specification `align = left` which requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 30).

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

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

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

Initial value: `empty`

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

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

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

For an example of customisation of the tabular notes, see p. 42.

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

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

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

13 Other features

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

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

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

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

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

13.2 Alignment option in `{NiceMatrix}`

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

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

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

13.3 The command `\rotate`

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

In the following command, we use that command in the `code-for-first-row`.³⁵

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

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

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

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

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

13.4 The option `small`

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

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

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

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

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

- the cells of the array are composed with `\scriptstyle`;

³⁵It can also be used in `RowStyle` (cf. p. 16).

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

13.5 The counters `iRow` and `jCol`

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column³⁶. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 12) and in the `\CodeAfter` (cf. p. 25), `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).

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

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

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

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

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

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

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

13.6 The option `light-syntax`

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

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

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

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

³⁶We 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.

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.³⁷

13.7 Color of the delimiters

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

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

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

This colour also applies to the delimiters drawn by the command `\SubMatrix` (cf. p. 26).

13.8 The environment `{NiceArrayWithDelims}`

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

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

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

14 Use of Tikz with nicematrix

14.1 The nodes corresponding to the contents of the cells

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

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

However, it's possible to impose the creation of a node with the command `\NotEmpty`.³⁸

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

³⁷The 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.

³⁸One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 20) and the computation of the “corners” (cf. p. 10).

PGF). However, one should remind that `nicematrix` doesn't load `Tikz` by default. In the following examples, we assume that `Tikz` has been loaded.

```

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

```

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

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

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

```

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

```

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

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 48).

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

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

In fact, the package `nicematrix` can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option `create-medium-nodes` and the second ones with the option `create-large-nodes`.³⁹

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.⁴⁰

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

The “large nodes” of the first column and last column may appear too small for some usage. That's why it's possible to use the options `left-margin` and `right-margin` to add space on both sides of

³⁹There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

⁴⁰There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 18).

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`.⁴¹

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

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

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

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

Here is an array composed with the following code:

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

The nodes we have described are not available by default in the `\CodeBefore` (described p. 12). 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).

14.3 The nodes which indicate the position of the rules

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

⁴¹The 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.

		tulipe	lys
arum			violette mauve
muguet	dahlia		

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

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

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

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

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

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

14.4 The nodes corresponding to the command `\SubMatrix`

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 26.

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}\}`).

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

15 API for the developpers

The package `nicematrix` provides two variables which are internal but public⁴²:

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

These variables contain the code of what we have called the “code-before” (usually specified at the beginning of the environment with the syntax using the keywords `\CodeBefore` and `\Body`) and the “code-after” (usually specified at the end of the environment after the keyword `\CodeAfter`). The developer can use them to add code from a cell of the array (the affectation must be global, allowing to exit the cell, which is a TeX group).

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

Example : We want to write a command `\crossbox` to draw a cross in the current cell. This command will take in an optional argument between square brackets for a list of pairs *key-value* which will be given to Tikz before the drawing.

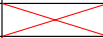
It’s possible to program such command `\crossbox` as follows, explicitly using the public variable `\g_nicematrix_code_after_tl`.

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

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

⁴²According 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.

16 Technical remarks

16.1 Definition of new column types

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

For example, one may wish to define a new column type `?` in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job⁴³:

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

The heavy vertical rule won't extend in the exterior rows.⁴⁴

```
\begin{pNiceArray}{cc?cc}[first-row,last-row=3]
```

```
C_1 & C_2 & C_3 & C_4 \\\
```

```
a & b & c & d \\\
```

```
e & f & g & h \\\
```

```
C_1 & C_2 & C_3 & C_4
```

```
\end{pNiceArray}$
```

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

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

16.2 Diagonal lines

By default, all the diagonal lines⁴⁵ 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):

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

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

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

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

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

The same example without parallelization:

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

⁴³The command `\vrule` is a TeX (and not LaTeX) command.

⁴⁴Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

⁴⁵We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in the `\CodeAfter`.

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

16.3 The “empty” cells

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

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

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

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

- Each cell whose TeX output has a width equal to zero is empty.
- A cell containing the command `\NotEmpty` is not empty (and a PGF/Tikz node) is created in that cell.
- A cell with a command `\Hspace` (or `\Hspace*`) is empty. This command `\Hspace` is a command defined by the package `nicematrix` with the same meaning as `\hspace` except that the cell where it is used is considered as empty. This command can be used to fix the width of some columns of the matrix without interfering with `nicematrix`.
- A cell of a column of type `p`, `m` or `t` is always considered as not empty. *Caution* : One should not rely upon that point because it may change if a future version of `nicematrix`.

16.4 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea⁴⁶. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`⁴⁷. The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` of `array` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

16.5 Incompatibilities

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

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

⁴⁶In 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).*

⁴⁷And 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.

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

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

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

17 Examples

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

```
\ttfamily \small
\begin{NiceTabular}{m{4.5cm}m{4.5cm}m{4.5cm}}[hvlines]
  \Block[tikz={pattern=grid,pattern color=lightgray}]{1}{1}
    {pattern = grid,\,\, pattern color = lightgray}
& \Block[tikz={pattern = north west lines,pattern color=blue}]{1}{1}
    {pattern = north west lines,\,\, pattern color = blue}
& \Block[tikz={outer color = red!50, inner color=white }]{2-1}{1}
    {outer color = red!50,\,\, inner color = white} \,\,
  \Block[tikz={pattern = sixpointed stars, pattern color = blue!15}]{1}{1}
    {pattern = sixpointed stars,\,\, pattern color = blue!15}
& \Block[tikz={left color = blue!50}]{1}{1}
    {left color = blue!50} \,\,
\end{NiceTabular}
```

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

17.2 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 12 p. 28.

Let’s consider that we wish to number the notes of a tabular with stars.⁴⁸

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument⁴⁹

```
\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}`.

⁴⁸Of course, it’s realistic only when there is very few notes in the tabular.

⁴⁹In fact: the value of its argument.

$$\left| \begin{array}{ccc} a_0 & & \\ & \ddots & \\ & & a_p \\ & & \vdots \\ & & a_p \end{array} \right. \begin{array}{ccc} & & a_0 \\ & & a_1 \\ & & \vdots \\ & & a_p \end{array} \left| \begin{array}{ccc} b_0 & & \\ & \ddots & \\ & & b_q \\ & & \vdots \\ & & b_q \end{array} \right|$$

An example for a linear system:

```

 $\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 0 \\ 0 & 1 & 0 & \cdots & 0 & \\ 0 & 0 & 1 & \ddots & \vdots & \\ & & & \ddots & \vdots & \\ \vdots & & & \ddots & 0 & \\ 0 & & & \cdots & 0 & 1 \end{pmatrix} \begin{pmatrix} \\ \\ \\ \\ \\ \\ \end{pmatrix} = \begin{pmatrix} 0 \\ L_2 - L_1 \\ L_3 - L_1 \\ \vdots \\ L_n - L_1 \end{pmatrix}$ 

```

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

17.4 Dotted lines which are no longer dotted

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

```

\NiceMatrixOptions{code-for-first-row = \scriptstyle,code-for-first-col = \scriptstyle }
\setcounter{MaxMatrixCols}{12}
\newcommand{\blue}{\color{blue}}
\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 0 \\ 0 & 1 & 0 & \cdots & 0 & \\ 0 & 0 & 1 & \ddots & \vdots & \\ \vdots & & & \ddots & \vdots & \\ 0 & \cdots & \cdots & 0 & 1 & \end{pmatrix} \begin{pmatrix} \\ \\ \\ \\ \\ \end{pmatrix} = \begin{pmatrix} 0 \\ L_2 \leftarrow L_2 - L_1 \\ L_3 \leftarrow L_3 - L_1 \\ \vdots \\ L_n \leftarrow L_n - L_1 \end{pmatrix}

```

$$\begin{pmatrix} 1 & \cdots & 1 \\ \cdots & 0 & 1 \\ \cdots & 1 & 0 \\ & & \ddots & \ddots \\ & & & 1 & \cdots & 1 \end{pmatrix} \begin{matrix} \\ \leftarrow i \\ \leftarrow j \\ \\ \\ \end{matrix}$$

$\begin{matrix} \uparrow \\ i \end{matrix}$
 $\begin{matrix} \uparrow \\ j \end{matrix}$

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

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

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

17.5 Stacks of matrices

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

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

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

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

```

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

```

```

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

```

```

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

```

```

\end{NiceMatrixBlock}

```

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

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

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

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

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}`

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{pmatrix} \\
 \begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{pmatrix} \begin{matrix} \\ L_2 \leftarrow L_1 - 4L_2 \\ L_3 \leftarrow L_1 + 4L_3 \\ L_4 \leftarrow 3L_1 - 4L_4 \end{matrix} \\
 \begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \\ \\ L_3 \leftarrow 3L_2 + L_3 \end{matrix} \\
 \begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \end{pmatrix}$$

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

```

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

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

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

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

17.6 How to highlight cells of a matrix

In order to highlight a cell of a matrix, it's possible to “draw” that cell with the key **draw** of the command `\Block` (this is one of the uses of a mono-cell block⁵⁰).

```

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

```

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

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.⁵¹

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}

```

⁵⁰We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

⁵¹For 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. We create a rectangular Tikz node which encompasses the nodes of the second row with the Tikz library `fit`. This Tikz node is filled after the construction of the matrix. In order to see the text *under* this node, we have to use transparency with the `blend mode` equal to `multiply`.

Caution : Some PDF readers are not able to show transparency.⁵²

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
```

⁵²In Overleaf, the “built-in” PDF viewer does not show transparency. You can switch to the “native” viewer in that case.

```

a & a + b & a + b + c & L_1 \\
a & a      & a + b      & L_2 \\
a & a      & a          & L_3
\end{pNiceArray}\]

```

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

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

```

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

```

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

17.7 Utilisation of \SubMatrix in the \CodeBefore

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

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

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

```

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

```

```

\[\begin{NiceArray}{*{6}{c}@{\hspace{6mm}}*{5}{c}}[nullify-dots]
\CodeBefore [create-cell-nodes]
  \SubMatrix({2-7}{6-11})
  \SubMatrix({7-2}{11-6})
  \SubMatrix({7-7}{11-11})
  \begin{tikzpicture}
    \node [highlight = (9-2) (9-6)] { } ;
    \node [highlight = (2-9) (6-9)] { } ;
  \end{tikzpicture}
\Body
  & & & & & & & \color{blue}\scriptstyle C_j \\\
  & & & & & & b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\\
  & & & & & & \Vdots & & \Vdots & & \Vdots \\\
  & & & & & & & & b_{kj} \\\
  & & & & & & & & \Vdots \\\
  & & & & & & b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn} \\\[3mm]
  & 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}\]

```

18 Implementation

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

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

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

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

Declaration of the package and packages loaded

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

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

<@@=nicematrix>

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

```

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

```

We give the traditional declaration of a package written with the L3 programming layer.

```

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

```

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

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

```

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

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

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

```

Technical definitions

```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

57 \cs_if_exist:NT \rvtx@ifformat@geq { \bool_set_true:N \c_@@_revtex_bool }

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

```

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

```

59 \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.

```

60 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
61   {
62     \iow_now:Nn \@mainaux
63     {
64       \ExplSyntaxOn
65       \cs_if_free:NT \pgfsyspdfmark
66         { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
67       \ExplSyntaxOff
68     }
69     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
70   }

```

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.

```

71 \ProvideDocumentCommand \iddots { }
72   {
73     \mathinner
74     {
75       \tex_mkern:D 1 mu
76       \box_move_up:nn { 1 pt } { \hbox:n { . } }
77       \tex_mkern:D 2 mu
78       \box_move_up:nn { 4 pt } { \hbox:n { . } }
79       \tex_mkern:D 2 mu
80       \box_move_up:nn { 7 pt }
81       { \vbox:n { \kern 7 pt \hbox:n { . } } }
82       \tex_mkern:D 1 mu
83     }
84   }

```

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.

```

85 \AtBeginDocument

```

```

86 {
87   \@ifpackageloaded { booktabs }
88   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
89   { }
90 }
91 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
92 {
93   \cs_set_eq:NN \@_old_pgfulil@check@rerun \pgfulil@check@rerun

```

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

```

94   \cs_set_protected:Npn \pgfulil@check@rerun ##1 ##2
95   {
96     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
97     { \@_old_pgfulil@check@rerun { ##1 } { ##2 } }
98   }
99 }

```

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

```

100 \bool_new:N \c_@@_colortbl_loaded_bool
101 \AtBeginDocument
102 {
103   \@ifpackageloaded { colortbl }
104   { \bool_set_true:N \c_@@_colortbl_loaded_bool }
105   {

```

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.

```

106     \cs_set_protected:Npn \CT@arc@ { }
107     \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
108     \cs_set:Npn \CT@arc@ #1 #2
109     {
110       \dim_compare:nNt \baselineskip = \c_zero_dim \noalign
111       { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
112     }

```

Idem for `\CT@drs@`.

```

113     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
114     \cs_set:Npn \CT@drs@ #1 #2
115     {
116       \dim_compare:nNt \baselineskip = \c_zero_dim \noalign
117       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
118     }
119     \cs_set:Npn \hline
120     {
121       \noalign { \ifnum 0 = ` } \fi
122       \cs_set_eq:NN \hskip \vskip
123       \cs_set_eq:NN \vrule \hrule
124       \cs_set_eq:NN \@width \@height
125       { \CT@arc@ \vline }
126       \futurelet \reserved@a
127       \@xhline
128     }
129   }
130 }

```

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

```

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

```

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

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

```

132 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
133 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
134 {
135   \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
136   \int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
137   \multispan { \int_eval:n { #2 - #1 + 1 } }
138   {
139     \CT@arc@
140     \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`⁵³

```

141   \skip_horizontal:N \c_zero_dim
142 }

```

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`.

```

143   \everycr { }
144   \cr
145   \noalign { \skip_vertical:N -\arrayrulewidth }
146 }

```

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.

```

147 \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`.

```

148 { \@@_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*.

```

149 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1-#2 \q_stop }
150 \cs_set:Npn \@@_cline_i:w #1-#2-#3 \q_stop
151 {

```

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).

```

152   \int_compare:nNnT { #1 } < { #2 }
153   { \multispan { \int_eval:n { #2 - #1 } } & }
154   \multispan { \int_eval:n { #3 - #2 + 1 } }
155   {
156     \CT@arc@
157     \leaders \hrule \@height \arrayrulewidth \hfill
158     \skip_horizontal:N \c_zero_dim
159   }

```

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

```

160   \peek_meaning_remove_ignore_spaces:NTF \cline
161   { & \@@_cline_i:en { \@@_succ:n { #3 } } }
162   { \everycr { } \cr }
163 }
164 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

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

```

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

```

The following command is a small shortcut.

```

167 \cs_new:Npn \@@_math_toggle_token:

```

⁵³See question 99041 on TeX StackExchange.

```

168 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

169 \cs_new_protected:Npn \@@_set_CT@arc@:
170 { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
171 \cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
172 { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
173 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
174 { \cs_set:Npn \CT@arc@ { \color { #1 } } }

175 \cs_set_eq:NN \@@_old_pgfpintanchor \pgfpintanchor

```

The column S of siunitx

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

```

176 \bool_new:N \c_@@_siunitx_loaded_bool
177 \AtBeginDocument
178 {
179   \ifpackageloaded { siunitx }
180     { \bool_set_true:N \c_@@_siunitx_loaded_bool }
181     { }
182 }

```

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

```

183 \AtBeginDocument
184 {
185   \bool_if:NTF { ! \c_@@_siunitx_loaded_bool }
186     { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
187     {

```

For version of siunitx at least equal to 3.0, the adaptation is different from previous ones. We test the version of siunitx by the existence of the control sequence \siunitx_cell_begin:w.

```

188   \cs_if_exist:NTF \siunitx_cell_begin:w
189   {
190     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
191     {
192       \renewcommand*{\NC@rewrite@S}[1] []
193       {
194         \@temptokena \exp_after:wN
195         {
196           \tex_the:D \@temptokena
197           > {
198             \@@_Cell:
199             \keys_set:nn { siunitx } { ##1 }
200             \siunitx_cell_begin:w
201           }

```

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

```

202       \@@_true_c:
203       < { \siunitx_cell_end: \@@_end_Cell: }
204     }
205     \NC@find
206   }
207 }
208 {
209   \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
210   {
211     \renewcommand*{\NC@rewrite@S}[1] []
212     {
213       \@temptokena \exp_after:wN
214       {

```



```

216             \tex_the:D \@temptokena
217             > { \c_@@_table_collect_begin_tl S {##1} }
218             \c_@@_true_c:
219             < { \c_@@_table_print_tl \c_@@_end_Cell: }
220         }
221     \NC@find
222 }
223 }
224 }
225 }
226 }

```

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

```

227 \AtBeginDocument
228 {
229     \cs_set_eq:NN \c_@@_adapt_S_column: \prg_do_nothing:
230     \bool_lazy_and:nnT
231     { \c_@@_siunitx_loaded_bool }
232     { ! \cs_if_exist_p:N \siunitx_cell_begin:w }
233     {
234         \cs_set_protected:Npn \c_@@_adapt_S_column:
235         {
236             \group_begin:
237             \@temptokena = { }
238             \cs_set_eq:NN \NC@find \prg_do_nothing:
239             \NC@rewrite@S { }
240             \tl_gset:NV \g_tmpa_tl \@temptokena
241             \group_end:
242             \tl_new:N \c_@@_table_collect_begin_tl
243             \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
244             \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
245             \tl_new:N \c_@@_table_print_tl
246             \tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl { -1 } }
247             \cs_gset_eq:NN \c_@@_adapt_S_column: \prg_do_nothing:
248         }
249     }
250 }

```

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.

```

251 \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).

```

252 \cs_new:Npn \c_@@_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.

```

253 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
254 { \int_use:N \g_@@_env_int }

```

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

```

255 \cs_new_protected:Npn \c_@@_qpoint:n #1
256 { \pgfpointanchor { \c_@@_env: - #1 } { center } }

```

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

```
257 \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).

```
258 \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.).

```
259 \dim_new:N \l_@@_col_width_dim
260 \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.

```
261 \int_new:N \g_@@_row_total_int
262 \int_new:N \g_@@_col_total_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.

```
263 \str_new:N \l_@@_hpos_cell_str
264 \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`.

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

Idem pour the mono-row blocks.

```
266 \dim_new:N \g_@@_blocks_ht_dim
267 \dim_new:N \g_@@_blocks_dp_dim
```

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

```
268 \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.

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

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

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

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

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

```
271 \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.

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

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

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

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

```
274 \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.

```
275 \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 }`).

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

```
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 vlins 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_set:Nn \g_@@_com_or_env_str { environment }
```

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

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

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`).

```
295 \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.

```
296 \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.

```
297 \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.

```
298 \int_new:N \l_@@_old_iRow_int
```

```
299 \int_new:N \l_@@_old_jCol_int
```

The TeX counters `\c@iRow` and `\c@jCol` will be created in the beginning of `{NiceArrayWithDelims}` (if they don't exist previously).

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

```
300 \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.

```
301 \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.

```
302 \bool_new:N \l_@@_X_columns_aux_bool
```

```
303 \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.

```
304 \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).

```
305 \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).

```
306 \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`).

```
307 \tl_new:N \l_@@_code_before_tl
308 \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).

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

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

```
310 \dim_new:N \l_@@_x_initial_dim
311 \dim_new:N \l_@@_y_initial_dim
312 \dim_new:N \l_@@_x_final_dim
313 \dim_new:N \l_@@_y_final_dim
```

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

```
314 \dim_zero_new:N \l_tmpc_dim
315 \dim_zero_new:N \l_tmpd_dim
```

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

```
316 \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”.

```
317 \dim_new:N \g_@@_width_last_col_dim
318 \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.

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

We also manage a sequence of the *positions* of the blocks. Of course, it's redundant with the previous sequence, but it's for efficiency. In that sequence, each block is represented by only the four first components: `{imin}{jmin}{imax}{jmax}`.

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

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

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

```
321 \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.

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

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

```
323 \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`).

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

The following flag will be raised if the key `width` has been raised. You use it to raise an error when this key is used while no column `X` is used.

```
325 \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) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

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

```
327 \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`).

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

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

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

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

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

```
332 \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).

```
333 \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`.

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

```
335 \tl_new:N \l_@@_draw_tl
```

```
336 \seq_new:N \l_@@_tikz_seq
```

```
337 \clist_new:N \l_@@_borders_clist
```

```
338 \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`.

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

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

```
340 \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`).

```
341 \str_new:N \l_@@_hpos_block_str
```

```
342 \str_set:Nn \l_@@_hpos_block_str { c }
```

```
343 \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`.

```
344 \tl_new:N \l_@@_vpos_of_block_tl
345 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

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

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

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

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

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

349 \dim_new:N \l_@@_submatrix_extra_height_dim
350 \dim_new:N \l_@@_submatrix_left_xshift_dim
351 \dim_new:N \l_@@_submatrix_right_xshift_dim
352 \clist_new:N \l_@@_hlines_clist
353 \clist_new:N \l_@@_vlines_clist
354 \clist_new:N \l_@@_submatrix_hlines_clist
355 \clist_new:N \l_@@_submatrix_vlines_clist
```

Variables for the exterior rows and columns

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

• First row

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

```
356 \int_new:N \l_@@_first_row_int
357 \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.

```
358 \int_new:N \l_@@_first_col_int
359 \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).

```
360 \int_new:N \l_@@_last_row_int
361 \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”.⁵⁴

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

⁵⁴We 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.

Idem for `\l_@@_last_col_without_value_bool`

```
363 \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.

```
364 \int_new:N \l_@@_last_col_int
365 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

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

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

The command `\tablarnote`

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

```
367 \newcounter { tablarnote }
```

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

```
368 \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.

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

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

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

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

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

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

```
372 \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`.

```
373 \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`.

```
374 \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`).

```
375 \AtBeginDocument
376 {
377   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
378   {
379     \NewDocumentCommand \tabularnote { m }
380     { \@@_error:n { enumitem-not-loaded } }
381   }
382   {
```

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.

```
383   \newlist { tabularnotes } { enumerate } { 1 }
384   \setlist [ tabularnotes ]
385   {
386     topsep = 0pt ,
387     noitemsep ,
388     leftmargin = * ,
389     align = left ,
390     labelsep = 0pt ,
391     label =
392     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
393   }
394   \newlist { tabularnotes* } { enumerate* } { 1 }
395   \setlist [ tabularnotes* ]
396   {
397     afterlabel = \nobreak ,
398     itemjoin = \quad ,
399     label =
400     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
401   }
```

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

Unfortunately, if the package `caption` is loaded, the command `\caption` evaluates its argument twice and since it is not aware (of course) of `\tabularnote`, the command `\tabularnote` is, in fact, not usable in `\caption` when `caption` is loaded.⁵⁵

```
402   \NewDocumentCommand \tabularnote { m }
403   {
404     \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
405     { \@@_error:n { tabularnote-forbidden } }
406     {
```

⁵⁵We should try to find a solution to that problem.

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

```
407 \int_incr:N \l_@@_number_of_notes_int
```

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

```
408 \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
409 \peek_meaning:NF \tabularnote
410 {
```

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.

```
411 \hbox_set:Nn \l_tmpa_box
412 {
```

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

```
413 \@@_notes_label_in_tabular:n
414 {
415 \stepcounter { tabularnote }
416 \@@_notes_style:n { tabularnote }
417 \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
418 {
419 ,
420 \stepcounter { tabularnote }
421 \@@_notes_style:n { tabularnote }
422 }
423 }
424 }
```

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.

```
425 \addtocounter { tabularnote } { -1 }
426 \refstepcounter { tabularnote }
427 \int_zero:N \l_@@_number_of_notes_int
428 \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).

```
429 \skip_horizontal:n { \box_wd:N \l_tmpa_box }
430 }
431 }
432 }
433 }
434 }
```

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.

```
435 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
436 {
437 \begin { pgfscope }
438 \pgfset
439 {
440 outer~sep = \c_zero_dim ,
441 inner~sep = \c_zero_dim ,
```

```

442     minimum~size = \c_zero_dim
443   }
444   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
445   \pgfnode
446   { rectangle }
447   { center }
448   {
449     \vbox_to_ht:nn
450     { \dim_abs:n { #5 - #3 } }
451     {
452       \vfill
453       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
454     }
455   }
456   { #1 }
457   { }
458   \end { pgfscope }
459 }

```

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

```

460 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
461 {
462   \begin { pgfscope }
463   \pgfset
464   {
465     outer~sep = \c_zero_dim ,
466     inner~sep = \c_zero_dim ,
467     minimum~size = \c_zero_dim
468   }
469   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
470   \pgfpointdiff { #3 } { #2 }
471   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
472   \pgfnode
473   { rectangle }
474   { center }
475   {
476     \vbox_to_ht:nn
477     { \dim_abs:n \l_tmpb_dim }
478     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
479   }
480   { #1 }
481   { }
482   \end { pgfscope }
483 }

```

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.

```

484 \bool_new:N \l_@@_colortbl_like_bool

```

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

```

485 \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`).

```
486 \dim_new:N \l_@@_cell_space_top_limit_dim
487 \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.

```
488 \dim_new:N \l_@@_inter_dots_dim
489 \AtBeginDocument { \dim_set:Nn \l_@@_inter_dots_dim { 0.45 em } }
```

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

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

```
490 \dim_new:N \l_@@_xdots_shorten_dim
491 \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 em } }
```

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

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

```
492 \dim_new:N \l_@@_radius_dim
493 \AtBeginDocument { \dim_set:Nn \l_@@_radius_dim { 0.53 pt } }
```

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

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

```
494 \tl_new:N \l_@@_xdots_line_style_tl
495 \tl_const:Nn \c_@@_standard_tl { standard }
496 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
497 \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).

```
498 \tl_new:N \l_@@_baseline_tl
499 \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`).

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

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

```
501 \bool_new:N \l_@@_parallelize_diags_bool
502 \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`.

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

```

504 \dim_new:N \l_@@_notes_above_space_dim
505 \AtBeginDocument { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }

```

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

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

```

506 \bool_new:N \l_@@_nullify_dots_bool

```

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).

```

507 \bool_new:N \l_@@_auto_columns_width_bool

```

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

```

508 \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.

```

509 \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”.

```

510 \bool_new:N \l_@@_medium_nodes_bool
511 \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).

```

512 \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`).

```

513 \dim_new:N \l_@@_left_margin_dim
514 \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`.

```

515 \dim_new:N \l_@@_extra_left_margin_dim
516 \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.

```

517 \tl_new:N \l_@@_end_of_row_tl
518 \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 “:”.

```

519 \tl_new:N \l_@@_xdots_color_tl

```

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

```

520 \tl_new:N \l_@@_delimiters_color_tl

```

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

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

```
522 \keys_define:nn { NiceMatrix / xdots }
523 {
524   line-style .code:n =
525   {
526     \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.

```
527     { \cs_if_exist_p:N \tikzpicture }
528     { \str_if_eq_p:nn { #1 } { standard } }
529     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
530     { \@@_error:n { bad-option-for~line-style } }
531   } ,
532   line-style .value_required:n = true ,
533   color .tl_set:N = \l_@@_xdots_color_tl ,
534   color .value_required:n = true ,
535   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
536   shorten .value_required:n = true ,
```

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

```
537   down .tl_set:N = \l_@@_xdots_down_tl ,
538   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).

```
539   draw-first .code:n = \prg_do_nothing: ,
540   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
541 }
```

```
542 \keys_define:nn { NiceMatrix / rules }
543 {
544   color .tl_set:N = \l_@@_rules_color_tl ,
545   color .value_required:n = true ,
546   width .dim_set:N = \arrayrulewidth ,
547   width .value_required:n = true
548 }
```

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

```
549 \keys_define:nn { NiceMatrix / Global }
550 {
551   delimiters .code:n =
552     \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
553   delimiters .value_required:n = true ,
554   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
555   rules .value_required:n = true ,
556   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
557   standard-cline .default:n = true ,
558   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
559   cell-space-top-limit .value_required:n = true ,
```

```

560 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
561 cell-space-bottom-limit .value_required:n = true ,
562 cell-space-limits .meta:n =
563 {
564     cell-space-top-limit = #1 ,
565     cell-space-bottom-limit = #1 ,
566 } ,
567 cell-space-limits .value_required:n = true ,
568 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
569 light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
570 light-syntax .default:n = true ,
571 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
572 end-of-row .value_required:n = true ,
573 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
574 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
575 last-row .int_set:N = \l_@@_last_row_int ,
576 last-row .default:n = -1 ,
577 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
578 code-for-first-col .value_required:n = true ,
579 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
580 code-for-last-col .value_required:n = true ,
581 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
582 code-for-first-row .value_required:n = true ,
583 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
584 code-for-last-row .value_required:n = true ,
585 hlines .clist_set:N = \l_@@_hlines_clist ,
586 vlines .clist_set:N = \l_@@_vlines_clist ,
587 hlines .default:n = all ,
588 vlines .default:n = all ,
589 vlines-in-sub-matrix .code:n =
590 {
591     \tl_if_single_token:nTF { #1 }
592     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
593     { \@@_error:n { One-letter-allowed } }
594 } ,
595 vlines-in-sub-matrix .value_required:n = true ,
596 hvlines .code:n =
597 {
598     \clist_set:Nn \l_@@_vlines_clist { all }
599     \clist_set:Nn \l_@@_hlines_clist { all }
600 } ,
601 hvlines-except-borders .code:n =
602 {
603     \clist_set:Nn \l_@@_vlines_clist { all }
604     \clist_set:Nn \l_@@_hlines_clist { all }
605     \bool_set_true:N \l_@@_except_borders_bool
606 } ,
607 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

With the option `renew-dots`, the commands `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

608 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
609 renew-dots .value_forbidden:n = true ,
610 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
611 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
612 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
613 create-extra-nodes .meta:n =
614 { create-medium-nodes , create-large-nodes } ,
615 left-margin .dim_set:N = \l_@@_left_margin_dim ,
616 left-margin .default:n = \arraycolsep ,
617 right-margin .dim_set:N = \l_@@_right_margin_dim ,
618 right-margin .default:n = \arraycolsep ,
619 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,

```

```

620 margin .default:n = \arraycolsep ,
621 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
622 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
623 extra-margin .meta:n =
624   { extra-left-margin = #1 , extra-right-margin = #1 } ,
625 extra-margin .value_required:n = true ,
626 }

```

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

```

627 \keys_define:nn { NiceMatrix / Env }
628 {

```

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

```

629   hvlines-except-corners .code:n =
630   {
631     \clist_set:Nn \l_@@_corners_clist { #1 }
632     \clist_set:Nn \l_@@_vlines_clist { all }
633     \clist_set:Nn \l_@@_hlines_clist { all }
634   } ,
635   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
636   corners .clist_set:N = \l_@@_corners_clist ,
637   corners .default:n = { NW , SW , NE , SE } ,
638   code-before .code:n =
639   {
640     \tl_if_empty:nF { #1 }
641     {
642       \tl_put_right:Nn \l_@@_code_before_tl { #1 }
643       \bool_set_true:N \l_@@_code_before_bool
644     }
645   } ,

```

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

```

646   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
647   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
648   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
649   baseline .tl_set:N = \l_@@_baseline_tl ,
650   baseline .value_required:n = true ,
651   columns-width .code:n =
652     \tl_if_eq:nnTF { #1 } { auto }
653     { \bool_set_true:N \l_@@_auto_columns_width_bool }
654     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
655   columns-width .value_required:n = true ,
656   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.

```

657   \legacy_if:nF { measuring@ }
658   {
659     \str_set:Nn \l_tmpa_str { #1 }
660     \seq_if_in:NVTf \g_@@_names_seq \l_tmpa_str
661     { \@@_error:nn { Duplicate-name } { #1 } }
662     { \seq_gput_left:N \g_@@_names_seq \l_tmpa_str }
663     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
664   } ,
665   name .value_required:n = true ,
666   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
667   code-after .value_required:n = true ,
668   colortbl-like .code:n =
669     \bool_set_true:N \l_@@_colortbl_like_bool
670     \bool_set_true:N \l_@@_code_before_bool ,
671   colortbl-like .value_forbidden:n = true
672 }

```



```

673 \keys_define:nn { NiceMatrix / notes }
674 {
675   para .bool_set:N = \l_@@_notes_para_bool ,
676   para .default:n = true ,
677   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
678   code-before .value_required:n = true ,
679   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
680   code-after .value_required:n = true ,
681   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
682   bottomrule .default:n = true ,
683   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
684   style .value_required:n = true ,
685   label-in-tabular .code:n =
686     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
687   label-in-tabular .value_required:n = true ,
688   label-in-list .code:n =
689     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
690   label-in-list .value_required:n = true ,
691   enumitem-keys .code:n =
692     {
693       \bool_if:NTF \c_@@_in_preamble_bool
694       {
695         \AtBeginDocument
696         {
697           \bool_if:NT \c_@@_enumitem_loaded_bool
698           { \setlist* [ tabularnotes ] { #1 } }
699         }
700       }
701       {
702         \bool_if:NT \c_@@_enumitem_loaded_bool
703         { \setlist* [ tabularnotes ] { #1 } }
704       }
705     } ,
706   enumitem-keys .value_required:n = true ,
707   enumitem-keys-para .code:n =
708     {
709       \bool_if:NTF \c_@@_in_preamble_bool
710       {
711         \AtBeginDocument
712         {
713           \bool_if:NT \c_@@_enumitem_loaded_bool
714           { \setlist* [ tabularnotes* ] { #1 } }
715         }
716       }
717       {
718         \bool_if:NT \c_@@_enumitem_loaded_bool
719         { \setlist* [ tabularnotes* ] { #1 } }
720       }
721     } ,
722   enumitem-keys-para .value_required:n = true ,
723   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
724 }
725 \keys_define:nn { NiceMatrix / delimiters }
726 {
727   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
728   max-width .default:n = true ,
729   color .tl_set:N = \l_@@_delimiters_color_tl ,
730   color .value_required:n = true ,
731 }

```

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

```

732 \keys_define:nn { NiceMatrix }
733 {
734   NiceMatrixOptions .inherit:n =
735     { NiceMatrix / Global } ,
736   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
737   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
738   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
739   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
740   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
741   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
742   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
743   NiceMatrix .inherit:n =
744     {
745       NiceMatrix / Global ,
746       NiceMatrix / Env ,
747     } ,
748   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
749   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
750   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
751   NiceTabular .inherit:n =
752     {
753       NiceMatrix / Global ,
754       NiceMatrix / Env
755     } ,
756   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
757   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
758   NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
759   NiceArray .inherit:n =
760     {
761       NiceMatrix / Global ,
762       NiceMatrix / Env ,
763     } ,
764   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
765   NiceArray / rules .inherit:n = NiceMatrix / rules ,
766   NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
767   pNiceArray .inherit:n =
768     {
769       NiceMatrix / Global ,
770       NiceMatrix / Env ,
771     } ,
772   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
773   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
774   pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
775 }

```

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

```

776 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
777 {
778   last-col .code:n = \tl_if_empty:nF { #1 }
779                 { \@@_error:n { last-col~non~empty~for~NiceMatrixOptions } }
780                 \int_zero:N \l_@@_last_col_int ,
781   small .bool_set:N = \l_@@_small_bool ,
782   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.

```

783   renew-matrix .code:n = \@@_renew_matrix: ,
784   renew-matrix .value_forbidden:n = true ,

```

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

```

785   transparent .code:n =
786     {
787       \@@_renew_matrix:

```

```

788     \bool_set_true:N \l_@@_renew_dots_bool
789     \@@_error:n { Key~transparent }
790   } ,
791   transparent .value_forbidden:n = true,

```

The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

```

792   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.

```

793   columns-width .code:n =
794     \tl_if_eq:nnTF { #1 } { auto }
795     { \@@_error:n { Option~auto~for~columns~width } }
796     { \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.

```

797   allow-duplicate-names .code:n =
798     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
799   allow-duplicate-names .value_forbidden:n = true ,

```

By default, the specifier used in the preamble of the array (for example in `{pNiceArray}`) to draw a vertical dotted line between two columns is the colon “:”. However, it’s possible to change this letter with `letter-for-dotted-lines` and, by the way, the letter “:” will remain free for other packages (for example `arydshln`).

```

800   letter-for-dotted-lines .code:n =
801   {
802     \tl_if_single_token:nTF { #1 }
803     { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
804     { \@@_error:n { One~letter~allowed } }
805   } ,
806   letter-for-dotted-lines .value_required:n = true ,
807   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
808   notes .value_required:n = true ,
809   sub-matrix .code:n =
810     \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
811   sub-matrix .value_required:n = true ,
812   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
813 }
814 \str_new:N \l_@@_letter_for_dotted_lines_str
815 \str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str

```

`\NiceMatrixOptions` is the command of the `nicematrix` package to fix options at the document level. The scope of these specifications is the current TeX group.

```

816 \NewDocumentCommand \NiceMatrixOptions { m }
817 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

818 \keys_define:nn { NiceMatrix / NiceMatrix }
819 {
820   last-col .code:n = \tl_if_empty:nTF {#1}
821   {
822     \bool_set_true:N \l_@@_last_col_without_value_bool
823     \int_set:Nn \l_@@_last_col_int { -1 }
824   }

```

```

825         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
826     l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
827     r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
828     small .bool_set:N = \l_@@_small_bool ,
829     small .value_forbidden:n = true ,
830     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
831 }

```

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

```

832 \keys_define:nn { NiceMatrix / NiceArray }
833 {

```

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.

```

834     small .bool_set:N = \l_@@_small_bool ,
835     small .value_forbidden:n = true ,
836     last-col .code:n = \tl_if_empty:nF { #1 }
837         { \@@_error:n { last-col-non-empty-for-NiceArray } }
838         \int_zero:N \l_@@_last_col_int ,
839     notes / para .bool_set:N = \l_@@_notes_para_bool ,
840     notes / para .default:n = true ,
841     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
842     notes / bottomrule .default:n = true ,
843     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
844     tabularnote .value_required:n = true ,
845     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
846     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
847     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
848 }
849 \keys_define:nn { NiceMatrix / pNiceArray }
850 {
851     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
852     last-col .code:n = \tl_if_empty:nF { #1 }
853         { \@@_error:n { last-col-non-empty-for-NiceArray } }
854         \int_zero:N \l_@@_last_col_int ,
855     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
856     small .bool_set:N = \l_@@_small_bool ,
857     small .value_forbidden:n = true ,
858     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
859     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
860     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
861 }

```

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

```

862 \keys_define:nn { NiceMatrix / NiceTabular }
863 {

```

The dimension width will be used if at least a column of type X is used.

```

864     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
865         \bool_set_true:N \l_@@_width_used_bool ,
866     width .value_required:n = true ,
867     notes / para .bool_set:N = \l_@@_notes_para_bool ,
868     notes / para .default:n = true ,
869     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
870     notes / bottomrule .default:n = true ,
871     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
872     tabularnote .value_required:n = true ,
873     last-col .code:n = \tl_if_empty:nF { #1 }
874         { \@@_error:n { last-col-non-empty-for-NiceArray } }

```

```

875             \int_zero:N \l_@@_last_col_int ,
876     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
877     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
878     unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
879 }

```

Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_Cell:-\@@_end_Cell:` will be used to format the cells of the array. In the code, the affectations are global because this pseudo-environment will be used in the cells of a `\halign` (via an environment `{array}`).

```

880 \cs_new_protected:Npn \@@_Cell:
881 {

```

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*).

```

882     \tl_gclear:N \g_@@_post_action_cell_tl

```

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

```

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

```

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

```

884     \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.

```

885     \int_compare:nNnT \c@jCol = 1
886     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }

```

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

```

887     \hbox_set:Nw \l_@@_cell_box
888     \bool_if:NF \l_@@_NiceTabular_bool
889     {
890         \c_math_toggle_token
891         \bool_if:NT \l_@@_small_bool \scriptstyle
892     }

```

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.

```

893     \g_@@_row_style_tl
894     \int_compare:nNnTF \c@iRow = 0
895     {
896         \int_compare:nNnT \c@jCol > 0
897         {
898             \l_@@_code_for_first_row_tl
899             \xglobal \colorlet { nicematrix-first-row } { . }
900         }
901     }
902     {
903         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
904         {
905             \l_@@_code_for_last_row_tl
906             \xglobal \colorlet { nicematrix-last-row } { . }
907         }
908     }
909 }

```

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.

```

910 \cs_new_protected:Npn \@@_begin_of_row:
911 {
912   \tl_gclear:N \g_@@_row_style_tl
913   \int_gincr:N \c@iRow
914   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
915   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
916   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
917   \pgfpicture
918   \pgfrememberpicturepositiononpagetrue
919   \pgfcoordinate
920   { \@@_env: - row - \int_use:N \c@iRow - base }
921   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
922   \str_if_empty:NF \l_@@_name_str
923   {
924     \pgfnodealias
925     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
926     { \@@_env: - row - \int_use:N \c@iRow - base }
927   }
928   \endpgfpicture
929 }

```

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.

```

930 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
931 {
932   \int_compare:nNnTF \c@iRow = 0
933   {
934     \dim_gset:Nn \g_@@_dp_row_zero_dim
935     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
936     \dim_gset:Nn \g_@@_ht_row_zero_dim
937     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
938   }
939   {
940     \int_compare:nNnT \c@iRow = 1
941     {
942       \dim_gset:Nn \g_@@_ht_row_one_dim
943       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
944     }
945   }
946 }
947 \cs_new_protected:Npn \@@_rotate_cell_box:
948 {
949   \box_rotate:Nn \l_@@_cell_box { 90 }
950   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
951   {
952     \vbox_set_top:Nn \l_@@_cell_box
953     {
954       \vbox_to_zero:n { }
955       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
956       \box_use:N \l_@@_cell_box
957     }
958   }
959   \bool_gset_false:N \g_@@_rotate_bool
960 }
961 \cs_new_protected:Npn \@@_adjust_size_box:
962 {

```

```

963 \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
964 {
965   \box_set_wd:Nn \l_@@_cell_box
966   { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
967   \dim_gzero:N \g_@@_blocks_wd_dim
968 }
969 \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
970 {
971   \box_set_dp:Nn \l_@@_cell_box
972   { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
973   \dim_gzero:N \g_@@_blocks_dp_dim
974 }
975 \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
976 {
977   \box_set_ht:Nn \l_@@_cell_box
978   { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
979   \dim_gzero:N \g_@@_blocks_ht_dim
980 }
981 }
982 \cs_new_protected:Npn \@@_end_Cell:
983 {
984   \@@_math_toggle_token:
985   \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.

```

986 \g_@@_post_action_cell_tl
987 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
988 \@@_adjust_size_box:
989 \box_set_ht:Nn \l_@@_cell_box
990 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
991 \box_set_dp:Nn \l_@@_cell_box
992 { \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”).

```

993 \dim_gset:Nn \g_@@_max_cell_width_dim
994 { \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”.

```

995 \@@_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.

```

996 \bool_if:NTF \g_@@_empty_cell_bool
997 { \box_use_drop:N \l_@@_cell_box }
998 {
999   \bool_lazy_or:nnTF
1000   \g_@@_not_empty_cell_bool
1001   { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1002   \@@_node_for_cell:
1003   { \box_use_drop:N \l_@@_cell_box }
1004 }
1005 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1006 \bool_gset_false:N \g_@@_empty_cell_bool
1007 \bool_gset_false:N \g_@@_not_empty_cell_bool
1008 }

```

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

```

1009 \cs_new_protected:Npn \@@_node_for_cell:
1010 {
1011   \pgfpicture
1012   \pgfsetbaseline \c_zero_dim
1013   \pgfrememberpicturepositiononpagetrue
1014   \pgfset
1015   {
1016     inner~sep = \c_zero_dim ,
1017     minimum~width = \c_zero_dim
1018   }
1019   \pgfnode
1020   { rectangle }
1021   { base }
1022   { \box_use_drop:N \l_@@_cell_box }
1023   { \@@_env: - \int_use:N \c_iRow - \int_use:N \c_jCol }
1024   { }
1025   \str_if_empty:NF \l_@@_name_str
1026   {
1027     \pgfnodealias
1028     { \l_@@_name_str - \int_use:N \c_iRow - \int_use:N \c_jCol }
1029     { \@@_env: - \int_use:N \c_iRow - \int_use:N \c_jCol }
1030   }
1031   \endpgfpicture
1032 }

```

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.

```

1033 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1034 {
1035   \cs_new_protected:Npn \@@_patch_node_for_cell:
1036   {
1037     \hbox_set:Nn \l_@@_cell_box
1038     {
1039       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1040       \hbox_overlap_left:n
1041       {
1042         \pgfsys@markposition
1043         { \@@_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.

```

1044       #1
1045     }
1046     \box_use:N \l_@@_cell_box
1047     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1048     \hbox_overlap_left:n

```



```

1049         {
1050             \pgfsys@markposition
1051             { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1052             #1
1053         }
1054     }
1055 }
1056 }

```

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

```

1057 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1058 {
1059     \@@_patch_node_for_cell:n
1060     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1061 }
1062 { \@@_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.

```

1063 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1064 {
1065     \bool_if:nTF { #1 } {\tl_gput_left:cx \tl_gput_right:cx
1066         { \g_@@_ #2 _ lines _ tl }
1067         {
1068             \use:c { @@ _ draw _ #2 : nnn }
1069             { \int_use:N \c@iRow }
1070             { \int_use:N \c@jCol }
1071             { \exp_not:n { #3 } }
1072         }
1073     }

```

We want to use `\array` of `array`. However, if the class used is `revtex4-1` or `revtex4-2`, we have to do some tuning and use the command `\@array@array` instead of `\array` because these classes do a redefinition of `\array` incompatible with our use of `\array`.

```

1074 \cs_new_protected:Npn \@@_revtex_array:
1075 {
1076     \cs_set_eq:NN \@acoll \@arrayacol
1077     \cs_set_eq:NN \@acolr \@arrayacol
1078     \cs_set_eq:NN \@acol \@arrayacol
1079     \cs_set_nopar:Npn \@halignto { }
1080     \@array@array
1081 }

```

```

1082 \cs_new_protected:Npn \l_@@_array:
1083 {
1084   \bool_if:NTF \c_@@_revtex_bool
1085     \l_@@_revtex_array:
1086   {
1087     \bool_if:NTF \l_@@_NiceTabular_bool
1088       { \dim_set_eq:NN \col@sep \tabcolsep }
1089       { \dim_set_eq:NN \col@sep \arraycolsep }
1090     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1091       { \cs_set_nopar:Npn \l_@@_halignto { } }
1092       { \cs_set_nopar:Npx \l_@@_halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1093   \l_@@_tabarray
1094 }

```

\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.

```

1095   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1096 }

```

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.

```

1097 \cs_set_eq:NN \l_@@_old_ialign: \ialign

```

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

```

1098 \cs_new_protected:Npn \l_@@_create_row_node:
1099 {

```

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

```

1100   \hbox
1101   {
1102     \bool_if:NT \l_@@_code_before_bool
1103     {
1104       \vtop
1105       {
1106         \skip_vertical:N 0.5\arrayrulewidth
1107         \pgfsys@markposition { \l_@@_env: - row - \l_@@_succ:n \c@iRow }
1108         \skip_vertical:N -0.5\arrayrulewidth
1109       }
1110     }
1111     \pgfpicture
1112     \pgfrememberpicturepositiononpagetrue
1113     \pgfcoordinate { \l_@@_env: - row - \l_@@_succ:n \c@iRow }
1114     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1115     \str_if_empty:NF \l_@@_name_str
1116     {
1117       \pgfnodealias
1118       { \l_@@_name_str - row - \l_@@_succ:n \c@iRow }
1119       { \l_@@_env: - row - \l_@@_succ:n \c@iRow }
1120     }
1121     \endpgfpicture
1122   }
1123 }

```

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

```

1124 \cs_new:Npn \l_@@_everycr: { \noalign { \l_@@_everycr_i: } }
1125 \cs_new_protected:Npn \l_@@_everycr_i:
1126 {
1127   \int_gzero:N \c@jCol
1128   \bool_gset_false:N \l_@@_after_col_zero_bool
1129   \bool_if:NF \l_@@_row_of_col_done_bool
1130   {
1131     \l_@@_create_row_node:

```

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

```

1132     \tl_if_empty:NF \l_@@_hlines_clist
1133     {
1134         \tl_if_eq:NnF \l_@@_hlines_clist { all }
1135         {
1136             \exp_args:NNx
1137             \clist_if_in:NnT
1138             \l_@@_hlines_clist
1139             { \@@_succ:n \c@iRow }
1140         }
1141     }

```

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.

```

1142     \int_compare:nNnT \c@iRow > { -1 }
1143     {
1144         \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@`.

```

1145         { \hrule height \arrayrulewidth width \c_zero_dim }
1146     }
1147 }
1148 }
1149 }
1150 }

```

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

```

1151 \cs_set_protected:Npn \@@_newcolumntype #1
1152 {
1153     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1154     \peek_meaning:NTF [
1155         { \newcol@ #1 }
1156         { \newcol@ #1 [ 0 ] }
1157     }

```

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

```

1158 \cs_set_protected:Npn \@@_renew_dots:
1159 {
1160     \cs_set_eq:NN \ldots \@@_Ldots
1161     \cs_set_eq:NN \cdots \@@_Cdots
1162     \cs_set_eq:NN \vdots \@@_Vdots
1163     \cs_set_eq:NN \ddots \@@_Ddots
1164     \cs_set_eq:NN \iddots \@@_Iddots
1165     \cs_set_eq:NN \dots \@@_Ldots
1166     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1167 }

```

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

```

1168 \cs_new_protected:Npn \@@_colortbl_like:
1169 {
1170     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1171     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1172     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1173 }

```

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

```
1174 \cs_new_protected:Npn \@@_pre_array_ii:
1175 {
```

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 ⁵⁶.

```
1176 \bool_if:NT \c_@@_booktabs_loaded_bool
1177 { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1178 \box_clear_new:N \l_@@_cell_box
1179 \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}`).

```
1180 \bool_if:NT \l_@@_small_bool
1181 {
1182     \cs_set_nopar:Npn \arraystretch { 0.47 }
1183     \dim_set:Nn \arraycolsep { 1.45 pt }
1184 }

1185 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1186 {
1187     \tl_put_right:Nn \@@_begin_of_row:
1188     {
1189         \pgfsys@markposition
1190         { \@@_env: - row - \int_use:N \c@iRow - base }
1191     }
1192 }
```

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`.

```
1193 \cs_set_nopar:Npn \ialign
1194 {
1195     \bool_if:NTF \c_@@_colortbl_loaded_bool
1196     {
1197         \CT@everycr
1198         {
1199             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1200             \@@_everycr:
1201         }
1202     }
1203     { \everycr { \@@_everycr: } }
1204     \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`⁵⁷ 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

⁵⁶cf. `\nicematrix@redefine@check@rerun`

⁵⁷The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

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`.

```

1205     \dim_gzero_new:N \g_@@_dp_row_zero_dim
1206     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1207     \dim_gzero_new:N \g_@@_ht_row_zero_dim
1208     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1209     \dim_gzero_new:N \g_@@_ht_row_one_dim
1210     \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1211     \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1212     \dim_gzero_new:N \g_@@_ht_last_row_dim
1213     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1214     \dim_gzero_new:N \g_@@_dp_last_row_dim
1215     \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`.

```

1216     \cs_set_eq:NN \ialign \@_old_ialign:
1217     \halign
1218 }

```

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).

```

1219     \cs_set_eq:NN \@_old_ldots \ldots
1220     \cs_set_eq:NN \@_old_cdots \cdots
1221     \cs_set_eq:NN \@_old_vdots \vdots
1222     \cs_set_eq:NN \@_old_ddots \ddots
1223     \cs_set_eq:NN \@_old_iddots \iddots
1224     \bool_if:NTF \l_@@_standard_cline_bool
1225     { \cs_set_eq:NN \cline \@_standard_cline }
1226     { \cs_set_eq:NN \cline \@_cline }
1227     \cs_set_eq:NN \Ldots \@_Ldots
1228     \cs_set_eq:NN \Cdots \@_Cdots
1229     \cs_set_eq:NN \Vdots \@_Vdots
1230     \cs_set_eq:NN \Ddots \@_Ddots
1231     \cs_set_eq:NN \Iddots \@_Iddots
1232     \cs_set_eq:NN \hdottedline \@_hdottedline:
1233     \cs_set_eq:NN \Hline \@_Hline:
1234     \cs_set_eq:NN \Hspace \@_Hspace:
1235     \cs_set_eq:NN \Hdotsfor \@_Hdotsfor:
1236     \cs_set_eq:NN \Vdotsfor \@_Vdotsfor:
1237     \cs_set_eq:NN \Block \@_Block:
1238     \cs_set_eq:NN \rotate \@_rotate:
1239     \cs_set_eq:NN \OnlyMainNiceMatrix \@_OnlyMainNiceMatrix:n
1240     \cs_set_eq:NN \dotfill \@_old_dotfill:
1241     \cs_set_eq:NN \CodeAfter \@_CodeAfter:
1242     \cs_set_eq:NN \diagbox \@_diagbox:nn
1243     \cs_set_eq:NN \NotEmpty \@_NotEmpty:
1244     \cs_set_eq:NN \RowStyle \@_RowStyle:n
1245     \bool_if:NT \l_@@_colortbl_like_bool \@_colortbl_like:
1246     \bool_if:NT \l_@@_renew_dots_bool \@_renew_dots:

```

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

```

1247     \cs_set_eq:NN \multicolumn \@_multicolumn:nnn
1248     \AtBeginEnvironment { tabular }
1249     { \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) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```
1250 \seq_gclear:N \g_@@_multicolumn_cells_seq
1251 \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).

```
1252 \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`).

```
1253 \int_gzero_new:N \g_@@_row_total_int
```

The counter `\c@jCol` will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter `\g_@@_col_total_int`. These counters are updated in the command `\@@_Cell`: executed at the beginning of each cell.

```
1254 \int_gzero_new:N \g_@@_col_total_int
1255 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1256 \@@_renew_NC@rewrite@S:
1257 \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.

```
1258 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1259 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1260 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1261 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1262 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1263 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1264 \tl_gclear_new:N \g_nicematrix_code_before_tl
1265 }
```

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

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

```
1266 \cs_new_protected:Npn \@@_pre_array:
1267 {
1268   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1269   \int_gzero_new:N \c@iRow
1270   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1271   \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).

```
1272 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1273 {
1274   \bool_set_true:N \l_@@_last_row_without_value_bool
1275   \bool_if:NT \g_@@_aux_found_bool
1276     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \c_@@_size_seq 3 } }
1277 }
1278 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1279 {
1280   \bool_if:NT \g_@@_aux_found_bool
1281     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \c_@@_size_seq 6 } }
1282 }
```

If there is a exterior row, we patch a command used in `\@@_Cell:` in order to keep track of some dimensions needed to the construction of that “last row”.

```

1283   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1284   {
1285     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1286     {
1287       \dim_gset:Nn \g_@@_ht_last_row_dim
1288       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1289       \dim_gset:Nn \g_@@_dp_last_row_dim
1290       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1291     }
1292   }

1293   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1294   \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.

```

1295   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the `aux` file.

```

1296   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1297   \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1298   \@@_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.

```

1299   \box_clear_new:N \l_@@_the_array_box

```

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

```

1300   \@@_construct_preamble:

```

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

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

```

1301   \dim_zero_new:N \l_@@_left_delim_dim
1302   \dim_zero_new:N \l_@@_right_delim_dim
1303   \bool_if:NTF \l_@@_NiceArray_bool
1304   {
1305     \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1306     \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1307   }
1308   {

```

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

```

1309   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1310   \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1311   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1312   \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1313   }

```

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).

```

1314   \hbox_set:Nw \l_@@_the_array_box

```

```

1315 \skip_horizontal:N \l_@@_left_margin_dim
1316 \skip_horizontal:N \l_@@_extra_left_margin_dim
1317 \c_math_toggle_token
1318 \bool_if:NTF \l_@@_light_syntax_bool
1319 { \use:c { @@-light-syntax } }
1320 { \use:c { @@-normal-syntax } }
1321 }

```

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

```

1322 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1323 {
1324   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1325   \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...

```

1326 \@@_pre_array:
1327 }

```

```

1328 \keys_define:nn { NiceMatrix / RowStyle }
1329 {
1330   cell-space-top-limit .code:n =
1331   {
1332     \tl_gput_right:Nn \g_@@_row_style_tl
1333     {
1334       \tl_gput_right:Nn \g_@@_post_action_cell_tl
1335       { \dim_set:Nn \l_@@_cell_space_top_limit_dim { #1 } }
1336     }
1337   } ,
1338   cell-space-top-limit .value_required:n = true ,
1339   cell-space-bottom-limit .code:n =
1340   {
1341     \tl_gput_right:Nn \g_@@_row_style_tl
1342     {
1343       \tl_gput_right:Nn \g_@@_post_action_cell_tl
1344       { \dim_set:Nn \l_@@_cell_space_bottom_limit_dim { #1 } }
1345     }
1346   } ,
1347   cell-space-bottom-limit .value_required:n = true ,
1348   cell-space-limits .meta:n =
1349   {
1350     cell-space-top-limit = #1 ,
1351     cell-space-bottom-limit = #1 ,
1352   } ,
1353   unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
1354 }

```

```

1355 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
1356 {
1357   \tl_gset:Nn \g_@@_row_style_tl { #2 }
1358   \keys_set:nn { NiceMatrix / RowStyle } { #1 }
1359   #2
1360   \ignorespaces
1361 }

```


The \CodeBefore

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

```
1362 \cs_new_protected:Npn \@@_pre_code_before:
1363 {
```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `code-before` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). 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).

```
1364 \int_set:Nn \c@iRow { \seq_item:Nn \c_@@_size_seq 2 }
1365 \int_set:Nn \c@jCol { \seq_item:Nn \c_@@_size_seq 5 }
1366 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \c_@@_size_seq 3 }
1367 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \c_@@_size_seq 6 }
```

Now, we will create all the `col` nodes and `row` nodes with the informations written in the `aux` file. You use the technique described in the page 1229 of `pgfmanual.pdf`, version 3.1.4b.

```
1368 \pgfsys@markposition { \@@_env: - position }
1369 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1370 \pgfpicture
1371 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1372 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1373 {
1374   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1375   \pgfcoordinate { \@@_env: - row - ##1 }
1376   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1377 }
```

Now, the recreation of the `col` nodes.

```
1378 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1379 {
1380   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1381   \pgfcoordinate { \@@_env: - col - ##1 }
1382   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1383 }
```

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

```
1384 \@@_create_diag_nodes:
```

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

```
1385 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1386 \endpgfpicture
1387 \bool_if:NT \c_@@_tikz_loaded_bool
1388 {
1389   \tikzset
1390   {
1391     every-picture / .style =
1392     { overlay , name-prefix = \@@_env: - }
1393   }
1394 }
1395 \cs_set_eq:NN \cellcolor \@@_cellcolor
1396 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1397 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1398 \cs_set_eq:NN \rowcolor \@@_rowcolor
1399 \cs_set_eq:NN \rowcolors \@@_rowcolors
1400 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1401 \cs_set_eq:NN \arraycolor \@@_arraycolor
1402 \cs_set_eq:NN \columncolor \@@_columncolor
1403 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1404 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1405 }
```

```

1406 \cs_new_protected:Npn \@@_exec_code_before:
1407 {
1408   \seq_gclear_new:N \g_@@_colors_seq
1409   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1410   \group_begin:

```

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

```

1411   \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`.

```

1412   \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.

```

1413   \@@_actually_color:
1414   \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1415   \group_end:
1416   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1417   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1418 }

```

```

1419 \keys_define:nn { NiceMatrix / CodeBefore }
1420 {
1421   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1422   create-cell-nodes .default:n = true ,
1423   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1424   sub-matrix .value_required:n = true ,
1425   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1426   delimiters / color .value_required:n = true ,
1427   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1428 }
1429 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1430 {
1431   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1432   \@@_CodeBefore:w
1433 }

```

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.

```

1434 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1435 {
1436   \bool_if:NT \g_@@_aux_found_bool
1437   {
1438     \@@_pre_code_before:
1439     #1
1440   }
1441 }

```

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.

```

1442 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1443 {

```

```

1444 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1445 {
1446   \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1447   \pgfcoordinate { \@@_env: - row - ##1 - base }
1448   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1449   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1450   {
1451     \cs_if_exist:cT
1452     { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1453     {
1454       \pgfsys@getposition
1455       { \@@_env: - ##1 - #####1 - NW }
1456       \@@_node_position:
1457       \pgfsys@getposition
1458       { \@@_env: - ##1 - #####1 - SE }
1459       \@@_node_position_i:
1460       \@@_pgf_rect_node:nnn
1461       { \@@_env: - ##1 - #####1 }
1462       { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1463       { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1464     }
1465   }
1466 }
1467 \@@_create_extra_nodes:
1468 }

```

The environment {NiceArrayWithDelims}

```

1469 \NewDocumentEnvironment { NiceArrayWithDelims }
1470 { m m O { } m ! O { } t \CodeBefore }
1471 {
1472   \@@_provide_pgfsyspdfmark:
1473   \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.

```

1474   \bgroup

1475   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1476   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1477   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1478   \int_gzero:N \g_@@_block_box_int
1479   \dim_zero:N \g_@@_width_last_col_dim
1480   \dim_zero:N \g_@@_width_first_col_dim
1481   \bool_gset_false:N \g_@@_row_of_col_done_bool
1482   \str_if_empty:NT \g_@@_name_env_str
1483   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }

```

The following line will be deleted when we will consider that only versions of `siunitx` after v3.0 are compatible with `nicematrix`.

```

1484   \@@_adapt_S_column:
1485   \bool_if:NTF \l_@@_NiceTabular_bool
1486     \mode_leave_vertical:
1487     \@@_test_if_math_mode:
1488     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1489     \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁵⁸. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able

⁵⁸e.g. `\color[rgb]{0.5,0.5,0}`

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.

```
1490 \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.

```
1491 \cs_if_exist:NT \tikz@library@external@loaded
1492 {
1493   \tikzexternaldisable
1494   \cs_if_exist:NT \ifstandalone
1495   { \tikzset { external / optimize = false } }
1496 }
```

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

```
1497 \int_gincr:N \g_@@_env_int
1498 \bool_if:NF \l_@@_block_auto_columns_width_bool
1499 { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

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

```
1500 \seq_gclear:N \g_@@_blocks_seq
1501 \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`.

```
1502 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1503 \seq_gclear:N \g_@@_pos_of_xdots_seq
1504 \tl_gclear_new:N \g_@@_code_before_tl
1505 \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.

```
1506 \bool_gset_false:N \g_@@_aux_found_bool
1507 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1508 {
1509   \bool_gset_true:N \g_@@_aux_found_bool
1510   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1511 }
```

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.

```
1512 \tl_gclear:N \g_@@_aux_tl
1513 \tl_if_empty:NF \g_@@_code_before_tl
1514 {
1515   \bool_set_true:N \l_@@_code_before_bool
1516   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1517 }
```

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`.

```
1518 \bool_if:NFT \l_@@_NiceArray_bool
1519 { \keys_set:nn { NiceMatrix / NiceArray } }
1520 { \keys_set:nn { NiceMatrix / pNiceArray } }
1521 { #3 , #5 }

1522 \tl_if_empty:NF \l_@@_rules_color_tl
1523 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
1524 % \bigskip
```

The argument `#6` is the last argument of `{NiceArrayWithDelims}`. With that argument of type “`t \CodeBefore`”, we test whether there is the keyword `\CodeBefore` at the beginning of

the environment. If that keyword is present, we have now to extract all the content between that keyword `\CodeBefore` and the (other) keyword `\Body`. It's the job that will do the command `\@@_pre_array_i:w`. After that job, the command `\@@_pre_array_i:w` will go on with `\@@_pre_array:`.

```

1525 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
1526 }
1527 {
1528   \bool_if:NTF \l_@@_light_syntax_bool
1529   { \use:c { end @@-light-syntax } }
1530   { \use:c { end @@-normal-syntax } }
1531   \c_math_toggle_token
1532   \skip_horizontal:N \l_@@_right_margin_dim
1533   \skip_horizontal:N \l_@@_extra_right_margin_dim
1534   \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.

```

1535 \bool_if:NT \l_@@_width_used_bool
1536 {
1537   \int_compare:nNnT \g_@@_total_X_weight_int = 0
1538   { \@@_error:n { width~without~X~columns } }
1539 }

```

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 .

```

1540 \int_compare:nNnT \g_@@_total_X_weight_int > 0
1541 {
1542   \bool_if:NTF \l_@@_X_columns_aux_bool
1543   { \dim_set_eq:NN \l_tmpa_dim \l_@@_X_columns_dim }
1544   {
1545     \dim_set:Nn \l_tmpa_dim
1546     {
1547       ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1548       / \int_use:N \g_@@_total_X_weight_int
1549     }
1550   }
1551   \tl_gput_right:Nx \g_@@_aux_tl
1552   {
1553     \bool_set_true:N \l_@@_X_columns_aux_bool
1554     \dim_set:Nn \l_@@_X_columns_dim { \dim_use:N \l_tmpa_dim }
1555   }
1556 }

```

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

```

1557 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1558 {
1559   \bool_if:NF \l_@@_last_row_without_value_bool
1560   {
1561     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1562     {
1563       \@@_error:n { Wrong~last~row }
1564       \int_gset_eq:NN \l_@@_last_row_int \c_iRow
1565     }
1566   }
1567 }

```

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”:⁵⁹

```

1568 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1569 \bool_if:nTF \g_@@_last_col_found_bool
1570 { \int_gdecr:N \c@jCol }
1571 {
1572   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1573   { \@@_error:n { last~col~not~used } }
1574 }

```

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

```

1575 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1576 \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. 119).

```

1577 \int_compare:nNnT \l_@@_first_col_int = 0
1578 {
1579   \skip_horizontal:N \col@sep
1580   \skip_horizontal:N \g_@@_width_first_col_dim
1581 }

```

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

```

1582 \bool_if:nTF \l_@@_NiceArray_bool
1583 {
1584   \str_case:VnF \l_@@_baseline_tl
1585   {
1586     b \@@_use_arraybox_with_notes_b:
1587     c \@@_use_arraybox_with_notes_c:
1588   }
1589   \@@_use_arraybox_with_notes:
1590 }

```

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).

```

1591 {
1592   \int_compare:nNnTF \l_@@_first_row_int = 0
1593   {
1594     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1595     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1596   }
1597   { \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”.⁶⁰

```

1598 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1599 {
1600   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1601   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1602 }
1603 { \dim_zero:N \l_tmpb_dim }
1604 \hbox_set:Nn \l_tmpa_box
1605 {
1606   \c_math_toggle_token
1607   \tl_if_empty:NF \l_@@_delimiters_color_tl
1608   { \color { \l_@@_delimiters_color_tl } }
1609   \exp_after:wN \left \g_@@_left_delim_tl
1610   \vcenter

```

⁵⁹We remind that the potential “first column” (exterior) has the number 0.

⁶⁰A 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).

```
1611 {
```

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.

```
1612 \skip_vertical:N -\l_tmpa_dim
1613 \hbox
1614 {
1615   \bool_if:NTF \l_@@_NiceTabular_bool
1616     { \skip_horizontal:N -\tabcolsep }
1617     { \skip_horizontal:N -\arraycolsep }
1618   \@@_use_arraybox_with_notes_c:
1619   \bool_if:NTF \l_@@_NiceTabular_bool
1620     { \skip_horizontal:N -\tabcolsep }
1621     { \skip_horizontal:N -\arraycolsep }
1622 }
```

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

```
1623 \skip_vertical:N -\l_tmpb_dim
1624 }
```

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.

```
1625 \tl_if_empty:NF \l_@@_delimiters_color_tl
1626 { \color { \l_@@_delimiters_color_tl } }
1627 \exp_after:wN \right \g_@@_right_delim_tl
1628 \c_math_toggle_token
1629 }
```

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.

```
1630 \bool_if:NTF \l_@@_delimiters_max_width_bool
1631 {
1632   \@@_put_box_in_flow_bis:nn
1633   \g_@@_left_delim_tl \g_@@_right_delim_tl
1634 }
1635 \@@_put_box_in_flow:
1636 }
```

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. 120).

```
1637 \bool_if:NT \g_@@_last_col_found_bool
1638 {
1639   \skip_horizontal:N \g_@@_width_last_col_dim
1640   \skip_horizontal:N \col@sep
1641 }
1642 \bool_if:NF \l_@@_Matrix_bool
1643 {
1644   \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1645   { \@@_error:n { columns~not~used } }
1646 }
1647 \group_begin:
1648 \globaldefs = 1
1649 \@@_msg_redirect_name:nn { columns~not~used } { error }
1650 \group_end:
1651 \@@_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.

```
1652 \egroup
```

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

```
1653 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1654 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1655 \iow_now:Nx \@mainaux
1656 {
```

```

1657      \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
1658      { \exp_not:V \g_@@_aux_tl }
1659    }
1660    \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1661    \bool_if:NT \c_@@_footnote_bool \endsavenotes
1662  }

```

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

We construct the preamble of the array

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

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

```

1663 \cs_new_protected:Npn \@@_construct_preamble:
1664 {

```

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

Since we use the tools of `array` to do this expansion, we will have a programming which is not in the style of the L3 programming layer.

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

```

1665   \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).

```

1666   \bool_if:NF \l_@@_Matrix_bool
1667   {
1668     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1669     \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by the L3 programming layer).

```

1670     \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1671     \@tempswatrue

```

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

```

1672     \@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`.

```

1673     \int_gzero:N \c@jCol
1674     \tl_gclear:N \g_@@_preamble_tl

```


`\g_tmpb_bool` will be raised if you have a `|` at the end of the preamble.

```

1675     \bool_gset_false:N \g_tmpb_bool
1676     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1677     {
1678         \tl_gset:Nn \g_@@_preamble_tl
1679         { ! { \skip_horizontal:N \arrayrulewidth } }
1680     }
1681     {
1682         \clist_if_in:NnT \l_@@_vlines_clist 1
1683         {
1684             \tl_gset:Nn \g_@@_preamble_tl
1685             { ! { \skip_horizontal:N \arrayrulewidth } }
1686         }
1687     }

```

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

```

1688     \int_gzero:N \g_@@_total_X_weight_int

```

The sequence `\g_@@_cols_vlsim_seq` will contain the numbers of the columns where you will to have to draw vertical lines in the potential sub-matrices (hence the name `vlism`).

```

1689     \seq_clear:N \g_@@_cols_vlism_seq

```

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

```

1690     \int_zero:N \l_tmpa_int

```

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

```

1691     \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1692     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1693 }

```

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

```

1694     \bool_if:NT \l_@@_colortbl_like_bool
1695     {
1696         \regex_replace_all:NnN
1697         \c_@@_columncolor_regex
1698         { \c { @@_columncolor_preamble } }
1699         \g_@@_preamble_tl
1700     }

```

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

```

1701     \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}`.

```

1702     \bool_lazy_or:nnT
1703     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1704     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1705     { \bool_set_false:N \l_@@_NiceArray_bool }

```

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

```

1706     \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).

```

1707     \int_compare:nNnTF \l_@@_first_col_int = 0
1708     { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1709     {
1710         \bool_lazy_all:nT
1711         {
1712             \l_@@_NiceArray_bool
1713             { \bool_not_p:n \l_@@_NiceTabular_bool }
1714             { \tl_if_empty_p:N \l_@@_vlines_clist }
1715             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }

```

```

1716     }
1717     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1718   }
1719   \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1720   { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1721   {
1722     \bool_lazy_all:nT
1723     {
1724       \l_@@_NiceArray_bool
1725       { \bool_not_p:n \l_@@_NiceTabular_bool }
1726       { \tl_if_empty_p:N \l_@@_vlines_clist }
1727       { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1728     }
1729     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1730   }

```

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`).

```

1731   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1732   {
1733     \tl_gput_right:Nn \g_@@_preamble_tl
1734     { > { \@@_error_too_much_cols: } 1 }
1735   }
1736 }

```

```

1737 \cs_new_protected:Npn \@@_patch_preamble:n #1
1738 {
1739   \str_case:nnF { #1 }
1740   {
1741     c { \@@_patch_preamble_i:n #1 }
1742     l { \@@_patch_preamble_i:n #1 }
1743     r { \@@_patch_preamble_i:n #1 }
1744     > { \@@_patch_preamble_ii:nn #1 }
1745     ! { \@@_patch_preamble_ii:nn #1 }
1746     @ { \@@_patch_preamble_ii:nn #1 }
1747     | { \@@_patch_preamble_iii:n #1 }
1748     p { \@@_patch_preamble_iv:n #1 }
1749     b { \@@_patch_preamble_iv:n #1 }
1750     m { \@@_patch_preamble_iv:n #1 }
1751     \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
1752     \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1753     \@@_true_c: { \@@_patch_preamble_vi:n #1 }
1754     ( { \@@_patch_preamble_vii:nn #1 }
1755     [ { \@@_patch_preamble_vii:nn #1 }
1756     \{ { \@@_patch_preamble_vii:nn #1 }
1757     ) { \@@_patch_preamble_viii:nn #1 }
1758     ] { \@@_patch_preamble_viii:nn #1 }
1759     \} { \@@_patch_preamble_viii:nn #1 }
1760     X { \@@_patch_preamble_ix:n }
1761     \q_stop { }
1762   }
1763   {
1764     \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1765     { \@@_patch_preamble_xi:n #1 }
1766     {
1767       \str_if_eq:VnTF \l_@@_letter_vlism_tl { #1 }
1768       {
1769         \seq_gput_right:Nx \g_@@_cols_vlism_seq
1770         { \int_eval:n { \c@jCol + 1 } }
1771         \tl_gput_right:Nx \g_@@_preamble_tl
1772         { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1773         \@@_patch_preamble:n

```

```

1774     }
1775     {
1776         \bool_lazy_and:nnTF
1777         { \str_if_eq_p:nn { : } { #1 } }
1778         \c_@@_arydshln_loaded_bool
1779         {
1780             \tl_gput_right:Nn \g_@@_preamble_tl { : }
1781             \@@_patch_preamble:n
1782         }
1783         { \@@_fatal:nn { unknown-column-type } { #1 } }
1784     }
1785 }
1786 }
1787 }

```

For c, l and r

```

1788 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1789 {
1790     \tl_gput_right:Nn \g_@@_preamble_tl
1791     {
1792         > { \@@_Cell: \str_set:Nn \l_@@_hpos_cell_str { #1 } }
1793         #1
1794         < \@@_end_Cell:
1795     }

```

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

```

1796     \int_gincr:N \c@jCol
1797     \@@_patch_preamble_x:n
1798 }

```

For >, ! and @

```

1799 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1800 {
1801     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1802     \@@_patch_preamble:n
1803 }

```

For |

```

1804 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1805 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1806     \int_incr:N \l_tmpa_int
1807     \@@_patch_preamble_iii_i:n
1808 }
1809 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1810 {
1811     \str_if_eq:nnTF { #1 } |
1812     { \@@_patch_preamble_iii:n | }
1813     {
1814         \tl_gput_right:Nx \g_@@_preamble_tl
1815         {
1816             \exp_not:N !
1817             {
1818                 \skip_horizontal:n
1819                 {
1820                     \dim_eval:n
1821                     {
1822                         \arrayrulewidth * \l_tmpa_int
1823                         + \doublerulesep * ( \l_tmpa_int - 1 )
1824                     }
1825                 }
1826             }

```

```

1827     }
1828     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1829     { \@@_vline:nn { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } }
1830     \int_zero:N \l_tmpa_int
1831     \str_if_eq:nnT { #1 } { \q_stop }
1832     { \bool_gset_true:N \g_tmpb_bool }
1833     \@@_patch_preamble:n #1
1834   }
1835 }
1836 \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.

```

1837 \keys_define:nn { WithArrows / p-column }
1838 {
1839   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
1840   r .value_forbidden:n = true ,
1841   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
1842   c .value_forbidden:n = true ,
1843   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
1844   l .value_forbidden:n = true ,
1845   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
1846   p .value_forbidden:n = true ,
1847   t .meta:n = p ,
1848   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
1849   m .value_forbidden:n = true ,
1850   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
1851   b .value_forbidden:n = true
1852 }

```

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

```

1853 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
1854 {
1855   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

1856   \@@_patch_preamble_iv_i:n
1857 }

1858 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
1859 {
1860   \str_if_eq:nnTF { #1 } { [ ]
1861     { \@@_patch_preamble_iv_ii:w [ ]
1862       { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
1863     }

1864   \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
1865   { \@@_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.

```

1866 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
1867 {

```

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).

```

1868   \str_set:Nn \l_@@_hpos_col_str { j }
1869   \keys_set:nn { WithArrows / p-column } { #1 }
1870   \@@_patch_preamble_iv_iv:n { #2 }
1871 }

```

The argument is the width of the column.

```

1872 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:n #1
1873 {
1874   \use:x
1875   {
1876     \@@_patch_preamble_iv_v:nnnn
1877     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
1878     { \dim_eval:n { #1 } }
1879     {

```

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

```

1880     \str_if_eq:VnTF \l_@@_hpos_col_str j
1881     { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
1882     { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str \l_@@_hpos_col_str }
1883     \str_case:Vn \l_@@_hpos_col_str
1884     {
1885       c { \exp_not:N \centering }
1886       l { \exp_not:N \raggedright }
1887       r { \exp_not:N \raggedleft }
1888     }
1889   }
1890   { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
1891 }

```

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

```

1892   \int_gincr:N \c@jCol
1893   \@@_patch_preamble_x:n
1894 }

```

#1 is the optional argument of `{minipage}`: `t` of `b`. Indeed, for the columns of type `m`, we use the value `b` here because there is a special post-action in order to center vertically the box (see **#4**).

#2 is the width of the `{minipage}`, that is to say also the width of the column.

#3 is the coding for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing).

#4 is an extra-code which contains `\l_@@_center_cell_box` (when the column is a `m` column) or nothing (in the other cases).

```

1895 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnn #1 #2 #3 #4
1896 {
1897   \tl_gput_right:Nn \g_@@_preamble_tl
1898   {
1899     > {

```

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.

```

1900     \dim_set:Nn \l_@@_col_width_dim { #2 }
1901     \@@_Cell:
1902     \begin { minipage } [ #1 ] { #2 }

```

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

```

1903     \everypar
1904     {
1905       \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
1906       \everypar { }
1907     }

```

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

```

1908     #3

```

The following code is to allow something like `\centering` in `\RowStyle`. However, there is extra vertical space if `\color{...}` is used in `\RowStyle` (we should try to add a key `color` to the command `\RowStyle`).

```

1909         \g_@@_row_style_tl
1910         \arraybackslash
1911     }
1912     c
1913     < {

```

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

```

1914         \@finalstrut \@arstrutbox
1915         \end { minipage }

```

If the letter in the preamble is `m`, `#3` will be equal to `\@@_center_cell_box:` (see just below).

```

1916         #4
1917         \@@_end_Cell:
1918     }
1919 }
1920 }

```

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.

```

1921 \cs_new_protected:Npn \@@_center_cell_box:
1922 {

```

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

```

1923     \tl_gput_right:Nn \g_@@_post_action_cell_tl
1924     {
1925         \int_compare:nNnT
1926         { \box_ht:N \l_@@_cell_box }
1927         >
1928         { \box_ht:N \@arstrutbox }
1929         {
1930             \hbox_set:Nn \l_@@_cell_box
1931             {
1932                 \box_move_down:nn
1933                 {
1934                     ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
1935                     + \baselineskip ) / 2
1936                 }
1937                 { \box_use:N \l_@@_cell_box }
1938             }
1939         }
1940     }
1941 }

```

For `w` and `W`

```

1942 \cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
1943 {
1944     \tl_gput_right:Nn \g_@@_preamble_tl
1945     {
1946         > {

```

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.

```

1947         \dim_set:Nn \l_@@_col_width_dim { #4 }
1948         \hbox_set:Nw \l_@@_cell_box
1949         \@@_Cell:
1950         \str_set:Nn \l_@@_hpos_cell_str { #3 }

```

```

1951     }
1952     c
1953     < {
1954         \@@_end_Cell:
1955         #1
1956         \hbox_set_end:
1957         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1958         \@@_adjust_size_box:
1959         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
1960     }
1961 }

```

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

```

1962     \int_gincr:N \c@jCol
1963     \@@_patch_preamble_x:n
1964 }

```

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

```

1965 \cs_new_protected:Npn \@@_patch_preamble_vi:n #1
1966 {
1967     \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

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

```

1968     \int_gincr:N \c@jCol
1969     \@@_patch_preamble_x:n
1970 }

```

For (, [and \{.

```

1971 \cs_new_protected:Npn \@@_patch_preamble_vii:nn #1 #2
1972 {
1973     \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.

```

1974     \int_compare:nNnTF \c@jCol = \c_zero_int
1975     {
1976         \str_if_eq:VnTF \g_@@_left_delim_tl { . }
1977         {

```

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

```

1978         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1979         \tl_gset:Nn \g_@@_right_delim_tl { . }
1980         \@@_patch_preamble:n #2
1981     }
1982     {
1983         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1984         \@@_patch_preamble_vii_i:nn { #1 } { #2 }
1985     }
1986 }
1987 { \@@_patch_preamble_vii_i:nn { #1 } { #2 } }
1988 }

```

```

1989 \cs_new_protected:Npn \@@_patch_preamble_vii_i:nn #1 #2
1990 {
1991     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1992     { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
1993     \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
1994     {
1995         \@@_error:nn { delimiter~after~opening } { #2 }
1996         \@@_patch_preamble:n
1997     }
1998     { \@@_patch_preamble:n #2 }
1999 }

```

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}`).

```

2000 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2001 {
2002   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter-with-small } }
2003   \tl_if_in:nnTF { ) ] \} } { #2 }
2004   { \@@_patch_preamble_viii_i:nnn #1 #2 }
2005   {
2006     \tl_if_eq:nnTF { \q_stop } { #2 }
2007     {
2008       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2009       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2010       {
2011         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2012         \tl_gput_right:Nx \g_@@_internal_code_after_tl
2013         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2014         \@@_patch_preamble:n #2
2015       }
2016     }
2017     {
2018       \tl_if_in:nnT { ( [ \{ } { #2 }
2019       { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2020       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2021       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2022       \@@_patch_preamble:n #2
2023     }
2024   }
2025 }

2026 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nnn #1 #2 #3
2027 {
2028   \tl_if_eq:nnTF { \q_stop } { #3 }
2029   {
2030     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2031     {
2032       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2033       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2034       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2035       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2036     }
2037     {
2038       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2039       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2040       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2041       \@@_error:nn { double-closing-delimiter } { #2 }
2042     }
2043   }
2044   {
2045     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2046     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2047     \@@_error:nn { double-closing-delimiter } { #2 }
2048     \@@_patch_preamble:n #3
2049   }
2050 }

```

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`.

```

2051 \cs_new_protected:Npn \@@_patch_preamble_ix:n #1
2052 {
2053   \str_if_eq:nnTF { #1 } { [ ]
2054   { \@@_patch_preamble_ix_i:w [ ]

```



```

2055     { \@@_patch_preamble_ix_i:w [ ] #1 }
2056   }

2057 \cs_new_protected:Npn \@@_patch_preamble_ix_i:w [ #1 ]
2058   { \@@_patch_preamble_ix_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 key 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).

```

2059 \keys_define:nn { WithArrows / X-column }
2060   { 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.

```

2061 \cs_new_protected:Npn \@@_patch_preamble_ix_ii:n #1
2062   {

```

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).

```

2063   \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).

```

2064   \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.

```

2065   \int_zero_new:N \l_@@_weight_int
2066   \int_set:Nn \l_@@_weight_int { 1 }
2067   \keys_set:known:nn { WithArrows / p-column } { #1 } \l_tmpa_tl
2068   \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2069   \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 on the aux file).

```

2070   \bool_if:NTF \l_@@_X_columns_aux_bool
2071     { \@@_patch_preamble_iv_iv:n { \l_@@_weight_int \l_@@_X_columns_dim } }
2072     {
2073       \tl_gput_right:Nn \g_@@_preamble_tl
2074         {
2075           > {
2076             \@@_Cell:
2077             \bool_set_true:N \l_@@_X_column_bool

```

The following code will nullify the box of the cell.

```

2078       \tl_gput_right:Nn \g_@@_post_action_cell_tl
2079         { \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.

```

2080       \begin { minipage } { 5 cm } \arraybackslash
2081     }
2082     c
2083     < {
2084       \end { minipage }
2085       \@@_end_Cell:
2086     }
2087   }
2088   \int_gincr:N \c@jCol
2089   \@@_patch_preamble_x:n
2090 }
2091 }

```

```

2092 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2093 {
2094   \tl_gput_right:Nn \g_@@_preamble_tl
2095   { ! { \skip_horizontal:N 2\l_@@_radius_dim } }

```

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

```

2096   \tl_gput_right:Nx \g_@@_internal_code_after_tl
2097   { \@@_vdottedline:n { \int_use:N \c@jCol } }
2098   \@@_patch_preamble:n
2099 }

```

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.

```

2100 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2101 {
2102   \str_if_eq:nnTF { #1 } { < }
2103   \@@_patch_preamble_xii:n
2104   {
2105     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2106     {
2107       \tl_gput_right:Nn \g_@@_preamble_tl
2108       { ! { \skip_horizontal:N \arrayrulewidth } }
2109     }
2110     {
2111       \exp_args:NNx
2112       \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2113       {
2114         \tl_gput_right:Nn \g_@@_preamble_tl
2115         { ! { \skip_horizontal:N \arrayrulewidth } }
2116       }
2117     }
2118     \@@_patch_preamble:n { #1 }
2119   }
2120 }
2121 \cs_new_protected:Npn \@@_patch_preamble_xii:n #1
2122 {
2123   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2124   \@@_patch_preamble_x:n
2125 }

```

The redefinition of `\multicolumn`

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

```

2126 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2127 {

```

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`.

```

2128   \multispan { #1 }
2129   \begingroup
2130   \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

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

```

2131   \@temptokena = { #2 }
2132   \@tempswatru
2133   \@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.

```
2134 \tl_gclear:N \g_@@_preamble_tl
2135 \exp_after:wN \@@_patch_m_preamble:n \the \temptokena \q_stop
```

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

```
2136 \exp_args:NV \mkpream \g_@@_preamble_tl
2137 \@addtopreamble \empty
2138 \endgroup
```

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

```
2139 \int_compare:nNnT { #1 } > 1
2140 {
2141   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2142   { \int_use:N \c@iRow - \@@_succ:n \c@jCol }
2143   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2144   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2145   {
2146     { \int_use:N \c@iRow }
2147     { \int_eval:n { \c@jCol + 1 } }
2148     { \int_use:N \c@iRow }
2149     { \int_eval:n { \c@jCol + #1 } }
2150   }
2151 }
```

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

```
2152 \cs_set:Npn \@sharp { #3 }
2153 \@arstrut
2154 \@preamble
2155 \null
```

We add some lines.

```
2156 \int_gadd:Nn \c@jCol { #1 - 1 }
2157 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2158 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2159 \ignorespaces
2160 }
```

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`.

```
2161 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2162 {
2163   \str_case:nnF { #1 }
2164   {
2165     c { \@@_patch_m_preamble_i:n #1 }
2166     l { \@@_patch_m_preamble_i:n #1 }
2167     r { \@@_patch_m_preamble_i:n #1 }
2168     > { \@@_patch_m_preamble_ii:nn #1 }
2169     ! { \@@_patch_m_preamble_ii:nn #1 }
2170     @ { \@@_patch_m_preamble_ii:nn #1 }
2171     | { \@@_patch_m_preamble_iii:n #1 }
2172     p { \@@_patch_m_preamble_iv:nnn t #1 }
2173     m { \@@_patch_m_preamble_iv:nnn c #1 }
2174     b { \@@_patch_m_preamble_iv:nnn b #1 }
2175     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2176     \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2177     \@@_true_c: { \@@_patch_m_preamble_vi:n #1 }
2178     \q_stop { }
2179   }
2180   { \@@_fatal:nn { unknown~column~type } { #1 } }
2181 }
```

For c, l and r

```

2182 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2183 {
2184   \tl_gput_right:Nn \g_@@_preamble_tl
2185   {
2186     > { \@@_Cell: \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2187     #1
2188     < \@@_end_Cell:
2189   }

```

We test for the presence of a <.

```

2190   \@@_patch_m_preamble_x:n
2191 }

```

For >, ! and @

```

2192 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2193 {
2194   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2195   \@@_patch_m_preamble:n
2196 }

```

For l

```

2197 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2198 {
2199   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2200   \@@_patch_m_preamble:n
2201 }

```

For p, m and b

```

2202 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2203 {
2204   \tl_gput_right:Nn \g_@@_preamble_tl
2205   {
2206     > {
2207       \@@_Cell:
2208       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2209       \mode_leave_vertical:
2210       \arraybackslash
2211       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2212     }
2213     c
2214     < {
2215       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2216       \end { minipage }
2217       \@@_end_Cell:
2218     }
2219   }

```

We test for the presence of a <.

```

2220   \@@_patch_m_preamble_x:n
2221 }

```

For w and W

```

2222 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2223 {
2224   \tl_gput_right:Nn \g_@@_preamble_tl
2225   {
2226     > {
2227       \hbox_set:Nw \l_@@_cell_box
2228       \@@_Cell:
2229       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2230     }
2231     c
2232     < {

```

```

2233         \@@_end_Cell:
2234         #1
2235         \hbox_set_end:
2236         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2237         \@@_adjust_size_box:
2238         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2239     }
2240 }

```

We test for the presence of a <.

```

2241     \@@_patch_m_preamble_x:n
2242 }

```

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

```

2243 \cs_new_protected:Npn \@@_patch_m_preamble_vi:n #1
2244 {
2245     \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

We test for the presence of a <.

```

2246     \@@_patch_m_preamble_x:n
2247 }

```

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.

```

2248 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2249 {
2250     \str_if_eq:nnTF { #1 } { < }
2251     \@@_patch_m_preamble_ix:n
2252     {
2253         \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2254         {
2255             \tl_gput_right:Nn \g_@@_preamble_tl
2256             { ! { \skip_horizontal:N \arrayrulewidth } }
2257         }
2258         {
2259             \exp_args:NNx
2260             \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2261             {
2262                 \tl_gput_right:Nn \g_@@_preamble_tl
2263                 { ! { \skip_horizontal:N \arrayrulewidth } }
2264             }
2265         }
2266         \@@_patch_m_preamble:n { #1 }
2267     }
2268 }
2269 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2270 {
2271     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2272     \@@_patch_m_preamble_x:n
2273 }

```

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).

```

2274 \cs_new_protected:Npn \@@_put_box_in_flow:
2275 {
2276     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2277     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2278     \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2279     { \box_use_drop:N \l_tmpa_box }
2280     \@@_put_box_in_flow_i:
2281 }

```

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).

```

2282 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2283 {
2284   \pgfpicture
2285     \@@_qpoint:n { row - 1 }
2286     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2287     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
2288     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2289     \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).

```

2290   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2291   {
2292     \int_set:Nn \l_tmpa_int
2293     {
2294       \str_range:Nnn
2295         \l_@@_baseline_tl
2296         6
2297         { \tl_count:V \l_@@_baseline_tl }
2298     }
2299     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2300   }
2301   {
2302     \str_case:VnF \l_@@_baseline_tl
2303     {
2304       { t } { \int_set:Nn \l_tmpa_int 1 }
2305       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2306     }
2307     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2308     \bool_lazy_or:nnT
2309     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2310     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2311     {
2312       \@@_error:n { bad~value~for~baseline }
2313       \int_set:Nn \l_tmpa_int 1
2314     }
2315     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2316     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2317   }
2318   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2319   \endpgfpicture
2320   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2321   \box_use_drop:N \l_tmpa_box
2322 }

```

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).

```

2323 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2324 {

```

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.

```

2325   \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2326   {
2327     \box_set_wd:Nn \l_@@_the_array_box
2328     { \box_wd:N \l_@@_the_array_box - \arraycolsep }

```

```
2329 }
```

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

```
2330 \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.

```
2331 \hbox:n
2332 {
2333 \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.

```
2334 \@@_create_extra_nodes:
2335 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2336 }
2337 \bool_lazy_or:nnT
2338 { \int_compare_p:nNn \c@tabularnote > 0 }
2339 { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2340 \@@_insert_tabularnotes:
2341 \end { minipage }
2342 }
```

```
2343 \cs_new_protected:Npn \@@_insert_tabularnotes:
2344 {
2345 \skip_vertical:N 0.65ex
```

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

```
2346 \group_begin:
2347 \l_@@_notes_code_before_tl
2348 \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.

```
2349 \int_compare:nNnT \c@tabularnote > 0
2350 {
2351 \bool_if:NTF \l_@@_notes_para_bool
2352 {
2353 \begin { tabularnotes* }
2354 \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2355 \end { tabularnotes* }
2356 }
```

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

```
2356 \par
2357 }
2358 {
2359 \tabularnotes
2360 \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2361 \endtabularnotes
2362 }
2363 }
2364 \unskip
2365 \group_end:
2366 \bool_if:NT \l_@@_notes_bottomrule_bool
2367 {
2368 \bool_if:NTF \c_@@_booktabs_loaded_bool
2369 {
```

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

```
2370 \skip_vertical:N \aboverulesep
```

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

```

2371      { \CT@arc@ \hrule height \heavyrulewidth }
2372    }
2373    { @@_error:n { bottomrule~without~booktabs } }
2374  }
2375  \l_@@_notes_code_after_tl
2376  \seq_gclear:N \g_@@_tabularnotes_seq
2377  \int_gzero:N \c@tabularnote
2378  }

```

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.

```

2379  \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2380  {
2381    \pgfpicture
2382      \@@_qpoint:n { row - 1 }
2383      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2384      \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2385      \dim_gsub:Nn \g_tmpa_dim \pgf@y
2386    \endpgfpicture
2387    \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2388    \int_compare:nNnT \l_@@_first_row_int = 0
2389    {
2390      \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2391      \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2392    }
2393    \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2394  }

```

Now, the general case.

```

2395  \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2396  {

```

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

```

2397    \tl_if_eq:NnT \l_@@_baseline_tl { t }
2398    { \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.

```

2399    \pgfpicture
2400      \@@_qpoint:n { row - 1 }
2401      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2402      \str_if_in:NnTF \l_@@_baseline_tl { line- }
2403      {
2404        \int_set:Nn \l_tmpa_int
2405        {
2406          \str_range:Nnn
2407            \l_@@_baseline_tl
2408            6
2409            { \tl_count:V \l_@@_baseline_tl }
2410        }
2411        \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2412      }
2413      {
2414        \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2415        \bool_lazy_or:nnT
2416          { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2417          { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2418          {
2419            \@@_error:n { bad~value~for~baseline }
2420            \int_set:Nn \l_tmpa_int 1
2421          }
2422        \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```



```

2423     }
2424     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2425     \endpgfpicture
2426     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2427     \int_compare:nNnT \l_@@_first_row_int = 0
2428     {
2429         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2430         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2431     }
2432     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2433 }

```

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.

```

2434 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2435 {

```

We will compute the real width of both delimiters used.

```

2436     \dim_zero_new:N \l_@@_real_left_delim_dim
2437     \dim_zero_new:N \l_@@_real_right_delim_dim
2438     \hbox_set:Nn \l_tmpb_box
2439     {
2440         \c_math_toggle_token
2441         \left #1
2442         \vcenter
2443         {
2444             \vbox_to_ht:nn

```

Here, you should use `\box_ht_plus_dp:N` when TeXLive 2021 will be available on Overleaf.

```

2445             { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2446             { }
2447         }
2448         \right .
2449         \c_math_toggle_token
2450     }
2451     \dim_set:Nn \l_@@_real_left_delim_dim
2452     { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
2453     \hbox_set:Nn \l_tmpb_box
2454     {
2455         \c_math_toggle_token
2456         \left .
2457         \vbox_to_ht:nn

```

Here, you should use `\box_ht_plus_dp:N` when TeXLive 2021 will be available on Overleaf.

```

2458             { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2459             { }
2460             \right #2
2461             \c_math_toggle_token
2462         }
2463     \dim_set:Nn \l_@@_real_right_delim_dim
2464     { \box_wd:N \l_tmpb_box - \nullldelimiterspace }

```

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

```

2465     \skip_horizontal:N \l_@@_left_delim_dim
2466     \skip_horizontal:N -\l_@@_real_left_delim_dim
2467     \@@_put_box_in_flow:
2468     \skip_horizontal:N \l_@@_right_delim_dim
2469     \skip_horizontal:N -\l_@@_real_right_delim_dim
2470 }

```

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).

```
2471 \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).

```
2472 {
2473   \peek_meaning_ignore_spaces:NTF \end \@@_analyze_end:Nn
```

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

```
2474   { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2475 }
2476 {
2477   \@@_create_col_nodes:
2478   \endarray
2479 }
```

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

```
2480 \NewDocumentEnvironment { @@-light-syntax } { b }
2481 {
```

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`.

```
2482   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2483   \tl_map_inline:nn { #1 }
2484   {
2485     \str_if_eq:nnT { ##1 } { & }
2486     { \@@_fatal:n { ampersand-in-light-syntax } }
2487     \str_if_eq:nnT { ##1 } { \ }
2488     { \@@_fatal:n { double-backslash-in-light-syntax } }
2489   }
```

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`.

```
2490   \@@_light_syntax_i #1 \CodeAfter \q_stop
2491 }
```

Now, the second part of the environment. It is empty. That's not surprising because we have caught the whole body of the environment with the specifier `b` provided by `xparse`.

```
2492 { }
2493 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2494 {
2495   \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).

```
2496   \seq_gclear_new:N \g_@@_rows_seq
2497   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2498   \exp_args:NNV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to know that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```
2499   \int_compare:nNnT \l_@@_last_row_int = { -1 }
2500   { \int_set:Nn \l_@@_last_row_int { \seq_count:N \g_@@_rows_seq } }
```

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

```

2501 \exp_args:NV \@@_array: \g_@@_preamble_tl
We need a global affectation because, when executing \l_tmpa_tl, we will exit the first cell of the
array.
2502 \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2503 \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
2504 \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2505 \@@_create_col_nodes:
2506 \endarray
2507 }
2508 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2509 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
2510 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2511 {
2512 \seq_gclear_new:N \g_@@_cells_seq
2513 \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2514 \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2515 \l_tmpa_tl
2516 \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2517 }

```

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).

```

2518 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2519 {
2520 \str_if_eq:VnT \g_@@_name_env_str { #2 }
2521 { \@@_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.

```

2522 \end { #2 }
2523 }

```

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).

```

2524 \cs_new:Npn \@@_create_col_nodes:
2525 {
2526 \crrc
2527 \int_compare:nNnT \l_@@_first_col_int = 0
2528 {
2529 \omit
2530 \hbox_overlap_left:n
2531 {
2532 \bool_if:NT \l_@@_code_before_bool
2533 { \pgfsys@markposition { \@@_env: - col - 0 } }
2534 \pgfpicture
2535 \pgfrememberpicturepositiononpagetrue
2536 \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
2537 \str_if_empty:NF \l_@@_name_str
2538 { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2539 \endpgfpicture
2540 \skip_horizontal:N 2\col@sep
2541 \skip_horizontal:N \g_@@_width_first_col_dim
2542 }
2543 &
2544 }
2545 \omit

```

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

```
2546 \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`).

```
2547 \int_compare:nNnTF \l_@@_first_col_int = 0
2548 {
2549   \bool_if:NT \l_@@_code_before_bool
2550   {
2551     \hbox
2552     {
2553       \skip_horizontal:N -0.5\arrayrulewidth
2554       \pgfsys@markposition { \@@_env: - col - 1 }
2555       \skip_horizontal:N 0.5\arrayrulewidth
2556     }
2557   }
2558   \pgfpicture
2559   \pgfrememberpicturepositiononpagetrue
2560   \pgfcoordinate { \@@_env: - col - 1 }
2561   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2562   \str_if_empty:NF \l_@@_name_str
2563   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2564   \endpgfpicture
2565 }
2566 {
2567   \bool_if:NT \l_@@_code_before_bool
2568   {
2569     \hbox
2570     {
2571       \skip_horizontal:N 0.5\arrayrulewidth
2572       \pgfsys@markposition { \@@_env: - col - 1 }
2573       \skip_horizontal:N -0.5\arrayrulewidth
2574     }
2575   }
2576   \pgfpicture
2577   \pgfrememberpicturepositiononpagetrue
2578   \pgfcoordinate { \@@_env: - col - 1 }
2579   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
2580   \str_if_empty:NF \l_@@_name_str
2581   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2582   \endpgfpicture
2583 }
```

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.

```
2584 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
2585 \bool_if:NF \l_@@_auto_columns_width_bool
2586 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2587 {
2588   \bool_lazy_and:nnTF
2589   \l_@@_auto_columns_width_bool
2590   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2591   { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2592   { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2593   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2594 }
2595 \skip_horizontal:N \g_tmpa_skip
2596 \hbox
2597 {
```

```

2598 \bool_if:NT \l_@@_code_before_bool
2599 {
2600     \hbox
2601     {
2602         \skip_horizontal:N -0.5\arrayrulewidth
2603         \pgfsys@markposition { \@@_env: - col - 2 }
2604         \skip_horizontal:N 0.5\arrayrulewidth
2605     }
2606 }
2607 \pgfpicture
2608 \pgfrememberpicturepositiononpagetrue
2609 \pgfcoordinate { \@@_env: - col - 2 }
2610 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2611 \str_if_empty:NF \l_@@_name_str
2612 { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2613 \endpgfpicture
2614 }

```

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.

```

2615 \int_gset:Nn \g_tmpa_int 1
2616 \bool_if:NTF \g_@@_last_col_found_bool
2617 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2618 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2619 {
2620     &
2621     \omit
2622     \int_gincr:N \g_tmpa_int

```

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

```

2623 \skip_horizontal:N \g_tmpa_skip
2624 \bool_if:NT \l_@@_code_before_bool
2625 {
2626     \hbox
2627     {
2628         \skip_horizontal:N -0.5\arrayrulewidth
2629         \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2630         \skip_horizontal:N 0.5\arrayrulewidth
2631     }
2632 }

```

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

```

2633 \pgfpicture
2634 \pgfrememberpicturepositiononpagetrue
2635 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2636 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2637 \str_if_empty:NF \l_@@_name_str
2638 {
2639     \pgfnodealias
2640     { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2641     { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2642 }
2643 \endpgfpicture
2644 }

2645 &
2646 \omit
2647 \int_gincr:N \g_tmpa_int
2648 \skip_horizontal:N \g_tmpa_skip
2649 \bool_lazy_all:nT
2650 {
2651     \l_@@_NiceArray_bool
2652     { \bool_not_p:n \l_@@_NiceTabular_bool }
2653     { \clist_if_empty_p:N \l_@@_vlines_clist }

```

```

2654         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2655         { ! \l_@@_bar_at_end_of_pream_bool }
2656     }
2657     { \skip_horizontal:N -\col@sep }
2658     \bool_if:NT \l_@@_code_before_bool
2659     {
2660         \hbox
2661         {
2662             \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.

```

2663         \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2664         { \skip_horizontal:N -\arraycolsep }
2665         \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2666         \skip_horizontal:N 0.5\arrayrulewidth
2667         \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2668         { \skip_horizontal:N \arraycolsep }
2669     }
2670 }
2671 \pgfpicture
2672 \pgfrememberpicturepositiononpagetrue
2673 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2674 {
2675     \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
2676     {
2677         \pgfpoint
2678         { - 0.5 \arrayrulewidth - \arraycolsep }
2679         \c_zero_dim
2680     }
2681     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2682 }
2683 \str_if_empty:NF \l_@@_name_str
2684 {
2685     \pgfnodealias
2686     { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2687     { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2688 }
2689 \endpgfpicture

2690 \bool_if:NT \g_@@_last_col_found_bool
2691 {
2692     \hbox_overlap_right:n
2693     {
2694         \skip_horizontal:N \g_@@_width_last_col_dim
2695         \bool_if:NT \l_@@_code_before_bool
2696         {
2697             \pgfsys@markposition
2698             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2699         }
2700         \pgfpicture
2701         \pgfrememberpicturepositiononpagetrue
2702         \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2703         \pgfpointorigin
2704         \str_if_empty:NF \l_@@_name_str
2705         {
2706             \pgfnodealias
2707             { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2708             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2709         }
2710         \endpgfpicture
2711     }

```

```

2712     }
2713     \cr
2714 }

```

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

```

2715 \tl_const:Nn \c_@@_preamble_first_col_tl
2716 {
2717   >
2718   {

```

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

```

2719     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
2720     \bool_gset_true:N \g_@@_after_col_zero_bool
2721     \@@_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.

```

2722     \hbox_set:Nw \l_@@_cell_box
2723     \@@_math_toggle_token:
2724     \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”.

```

2725     \bool_lazy_and:nnT
2726     { \int_compare_p:nNn \c@iRow > 0 }
2727     {
2728       \bool_lazy_or_p:nn
2729       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2730       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2731     }
2732     {
2733       \l_@@_code_for_first_col_tl
2734       \xglobal \colorlet { nicematrix-first-col } { . }
2735     }
2736 }

```

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`.

```

2737   l
2738   <
2739   {
2740     \@@_math_toggle_token:
2741     \hbox_set_end:
2742     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2743     \@@_adjust_size_box:
2744     \@@_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.

```

2745     \dim_gset:Nn \g_@@_width_first_col_dim
2746     { \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.

```

2747     \hbox_overlap_left:n
2748     {
2749       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2750       \@@_node_for_cell:
2751       { \box_use_drop:N \l_@@_cell_box }
2752       \skip_horizontal:N \l_@@_left_delim_dim
2753       \skip_horizontal:N \l_@@_left_margin_dim
2754       \skip_horizontal:N \l_@@_extra_left_margin_dim
2755     }
2756     \bool_gset_false:N \g_@@_empty_cell_bool

```

```

2757     \skip_horizontal:N -2\col@sep
2758   }
2759 }

```

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

```

2760 \tl_const:Nn \c_@@_preamble_last_col_tl
2761 {
2762   >
2763   {

```

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

```

2764     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n

```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```

2765     \bool_gset_true:N \g_@@_last_col_found_bool
2766     \int_gincr:N \c@jCol
2767     \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.

```

2768     \hbox_set:Nw \l_@@_cell_box
2769     \@@_math_toggle_token:
2770     \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”.

```

2771     \int_compare:nNnT \c@iRow > 0
2772     {
2773       \bool_lazy_or:nnT
2774       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2775       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2776       {
2777         \l_@@_code_for_last_col_tl
2778         \xglobal \colorlet { nicematrix-last-col } { . }
2779       }
2780     }
2781   }
2782   l
2783   <
2784   {
2785     \@@_math_toggle_token:
2786     \hbox_set_end:
2787     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2788     \@@_adjust_size_box:
2789     \@@_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.

```

2790     \dim_gset:Nn \g_@@_width_last_col_dim
2791     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2792     \skip_horizontal:N -2\col@sep

```

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

```

2793     \hbox_overlap_right:n
2794     {
2795       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2796       {
2797         \skip_horizontal:N \l_@@_right_delim_dim
2798         \skip_horizontal:N \l_@@_right_margin_dim
2799         \skip_horizontal:N \l_@@_extra_right_margin_dim
2800         \@@_node_for_cell:
2801       }
2802     }
2803     \bool_gset_false:N \g_@@_empty_cell_bool
2804   }
2805 }

```


The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}` but, in fact, there is a flag `\l_@@_NiceArray_bool`. In `{NiceArrayWithDelims}`, some special code will be executed if this flag is raised.

```
2806 \NewDocumentEnvironment { NiceArray } { }
2807 {
2808   \bool_set_true:N \l_@@_NiceArray_bool
2809   \str_if_empty:NT \g_@@_name_env_str
2810     { \str_gset:Nn \g_@@_name_env_str { NiceArray } }
```

We put `.` and `.` for the delimiters but, in fact, that doesn't matter because these arguments won't be used in `{NiceArrayWithDelims}` (because the flag `\l_@@_NiceArray_bool` is raised).

```
2811   \NiceArrayWithDelims . .
2812 }
2813 { \endNiceArrayWithDelims }
```

We create the variants of the environment `{NiceArrayWithDelims}`.

```
2814 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2815 {
2816   \NewDocumentEnvironment { #1 NiceArray } { }
2817   {
2818     \str_if_empty:NT \g_@@_name_env_str
2819       { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2820     \@@_test_if_math_mode:
2821     \NiceArrayWithDelims #2 #3
2822   }
2823   { \endNiceArrayWithDelims }
2824 }
2825 \@@_def_env:nnn p ( )
2826 \@@_def_env:nnn b [ ]
2827 \@@_def_env:nnn B \{ \}
2828 \@@_def_env:nnn v | |
2829 \@@_def_env:nnn V \| \|
```

The environment `{NiceMatrix}` and its variants

```
2830 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2831 {
2832   \bool_set_true:N \l_@@_Matrix_bool
2833   \use:c { #1 NiceArray }
2834   {
2835     *
2836     {
2837       \int_compare:nNnTF \l_@@_last_col_int < 0
2838         \c@MaxMatrixCols
2839         { \@@_pred:n \l_@@_last_col_int }
2840     }
2841     { > \@@_Cell: #2 < \@@_end_Cell: }
2842   }
2843 }
2844 \clist_map_inline:nn { { } , p , b , B , v , V }
2845 {
2846   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2847   {
2848     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2849     \tl_set:Nn \l_@@_type_of_col_tl c
2850     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2851     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2852   }
2853   { \use:c { end #1 NiceArray } }
```

```
2854 }
```

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

```
2855 \cs_new_protected:Npn \@@_NotEmpty:
2856 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

{NiceTabular}, {NiceTabularX} and {NiceTabular*}

```
2857 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2858 {
```

The dimension `\l_@@_width_dim` will be used if at least one column of type `X` is used.

```
2859 \dim_zero_new:N \l_@@_width_dim
2860 \dim_set_eq:NN \l_@@_width_dim \linewidth
2861 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2862 \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2863 \bool_set_true:N \l_@@_NiceTabular_bool
2864 \NiceArray { #2 }
2865 }
2866 { \endNiceArray }
```

```
2867 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
2868 {
2869 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
2870 \dim_zero_new:N \l_@@_width_dim
2871 \dim_set:NN \l_@@_width_dim { #1 }
2872 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2873 \bool_set_true:N \l_@@_NiceTabular_bool
2874 \NiceArray { #3 }
2875 }
2876 { \endNiceArray }
```

```
2877 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
2878 {
2879 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2880 \dim_set:NN \l_@@_tabular_width_dim { #1 }
2881 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2882 \bool_set_true:N \l_@@_NiceTabular_bool
2883 \NiceArray { #3 }
2884 }
2885 { \endNiceArray }
```

After the construction of the array

```
2886 \cs_new_protected:Npn \@@_after_array:
2887 {
2888 \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.

```
2889 \bool_if:NT \g_@@_last_col_found_bool
2890 { \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`.

```
2891 \bool_if:NT \l_@@_last_col_without_value_bool
2892 { \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.

```

2893 \bool_if:NT \l_@@_last_row_without_value_bool
2894 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

2895 \tl_gput_right:Nx \g_@@_aux_tl
2896 {
2897   \seq_gset_from_clist:Nn \exp_not:N \c_@@_size_seq
2898   {
2899     \int_use:N \l_@@_first_row_int ,
2900     \int_use:N \c_iRow ,
2901     \int_use:N \g_@@_row_total_int ,
2902     \int_use:N \l_@@_first_col_int ,
2903     \int_use:N \c_jCol ,
2904     \int_use:N \g_@@_col_total_int
2905   }
2906 }

```

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

```

2907 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
2908 {
2909   \tl_gput_right:Nx \g_@@_aux_tl
2910   {
2911     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
2912     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
2913   }
2914 }
2915 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
2916 {
2917   \tl_gput_right:Nx \g_@@_aux_tl
2918   {
2919     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
2920     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
2921     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
2922     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
2923   }
2924 }

```

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

```

2925 \@@_create_diag_nodes:

```

By default, the diagonal lines will be parallelized⁶¹. 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.

```

2926 \bool_if:NT \l_@@_parallelize_diags_bool
2927 {
2928   \int_gzero_new:N \g_@@_ddots_int
2929   \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.

```

2930   \dim_gzero_new:N \g_@@_delta_x_one_dim
2931   \dim_gzero_new:N \g_@@_delta_y_one_dim
2932   \dim_gzero_new:N \g_@@_delta_x_two_dim
2933   \dim_gzero_new:N \g_@@_delta_y_two_dim
2934 }
2935 \int_zero_new:N \l_@@_initial_i_int
2936 \int_zero_new:N \l_@@_initial_j_int
2937 \int_zero_new:N \l_@@_final_i_int
2938 \int_zero_new:N \l_@@_final_j_int

```

⁶¹It's possible to use the option `parallelize-diags` to disable this parallelization.

```

2939 \bool_set_false:N \l_@@_initial_open_bool
2940 \bool_set_false:N \l_@@_final_open_bool

```

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

```

2941 \bool_if:NT \l_@@_small_bool
2942 {
2943   \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
2944   \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }

```

The dimension `\l_@@_xdots_shorten_dim` corresponds to the option `xdots/shorten` available to the user. That’s why we give a new value according to the current value, and not an absolute value.

```

2945   \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
2946 }

```

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

```

2947 \@@_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.

```

2948 \@@_compute_corners:

```

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

```

2949 \@@_adjust_pos_of_blocks_seq:

```

The following code is only for efficiency. We determine whether the potential horizontal and vertical rules are “complete”, that is to say drawn in the whole array. We are sure that all the rules will be complete when there is no block, no virtual block (determined by a command such as `\Cdots`, `\Vdots`, etc.) and no corners. In that case, we switch to a shortcut version of `\@@_vline_i:nn` and `\@@_hline:nn`.

```

2950 \bool_lazy_all:nT
2951 {
2952   { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
2953   { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
2954   { \seq_if_empty_p:N \l_@@_corners_cells_seq }
2955 }
2956 {
2957   \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn
2958   \cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn
2959 }
2960 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
2961 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
2962 \cs_set_eq:NN \SubMatrix \@@_SubMatrix

```

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

```

2963 \bool_if:NT \c_@@_tikz_loaded_bool
2964 {
2965   \tikzset
2966   {
2967     every~picture / .style =
2968     {
2969       overlay ,
2970       remember~picture ,
2971       name~prefix = \@@_env: -
2972     }
2973   }
2974 }
2975 \cs_set_eq:NN \line \@@_line

```

```

2976 \g_@@_internal_code_after_tl
2977 \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.

```

2978 \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).

```

2979 \seq_gclear:N \g_@@_submatrix_names_seq

```

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

```

2980 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token

```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```

2981 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
2982 \scan_stop:
2983 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
2984 \tl_gclear:N \g_nicematrix_code_after_tl
2985 \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.

```

2986 \tl_if_empty:NF \g_nicematrix_code_before_tl
2987 {

```

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`.

```

2988 \cs_set_protected:Npn \rectanglecolor { }
2989 \cs_set_protected:Npn \columncolor { }
2990 \tl_gput_right:Nx \g_@@_aux_tl
2991 {
2992   \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
2993   { \exp_not:V \g_nicematrix_code_before_tl }
2994 }
2995 \bool_set_true:N \l_@@_code_before_bool
2996 }
2997 % \bool_if:NT \l_@@_code_before_bool \@@_write_aux_for_cell_nodes:

2998 \str_gclear:N \g_@@_name_env_str
2999 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁶². 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`.

```

3000 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3001 }

```

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`.

```

3002 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3003 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }

```

⁶²e.g. `\color[rgb]{0.5,0.5,0}`

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).

```

3004 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3005 {
3006   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3007   { \@@_adjust_pos_of_blocks_seq_i:nnnn #1 }
3008 }

```

The following command must *not* be protected.

```

3009 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnn #1 #2 #3 #4
3010 {
3011   { #1 }
3012   { #2 }
3013   {
3014     \int_compare:nNnTF { #3 } > { 99 }
3015     { \int_use:N \c@iRow }
3016     { #3 }
3017   }
3018   {
3019     \int_compare:nNnTF { #4 } > { 99 }
3020     { \int_use:N \c@jCol }
3021     { #4 }
3022   }
3023 }

```

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).

```

3024 \AtBeginDocument
3025 {
3026   \cs_new_protected:Npx \@@_draw_dotted_lines:
3027   {
3028     \c_@@_pgfortikzpicture_tl
3029     \@@_draw_dotted_lines_i:
3030     \c_@@_endpgfortikzpicture_tl
3031   }
3032 }

```

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

```

3033 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3034 {
3035   \pgfrememberpicturerepositiononpagetrue
3036   \pgf@relevantforpicturesizefalse
3037   \g_@@_HVdotsfor_lines_tl
3038   \g_@@_Vdots_lines_tl
3039   \g_@@_Ddots_lines_tl
3040   \g_@@_Iddots_lines_tl
3041   \g_@@_Cdots_lines_tl
3042   \g_@@_Ldots_lines_tl
3043 }

3044 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3045 {
3046   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3047   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3048 }

```

We define a new PGF shape for the diag nodes because we want to provide a anchor called .5 for those nodes.

```

3049 \pgfdeclareshape { @@_diag_node }
3050 {
3051   \savedanchor { \five }
3052   {
3053     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3054     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3055   }
3056   \anchor { 5 } { \five }
3057   \anchor { center } { \pgfpointorigin }
3058 }

3059 \cs_new_protected:Npn @@_write_aux_for_cell_nodes:
3060 {
3061   \pgfpicture
3062   \pgfrememberpicturepositiononpagetrue
3063   \pgf@relevantforpicturesizefalse
3064   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3065   {
3066     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3067     {
3068       \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
3069       {
3070         \pgfscope
3071         \pgftransformshift
3072         { \pgfpointanchor { \@@_env: - ##1 - #####1 } { north-west } }
3073         \pgfnode
3074         { rectangle }
3075         { center }
3076         {
3077           \hbox
3078             { \pgfsys@markposition { \@@_env: - ##1 - #####1 - NW } }
3079           }
3080         { }
3081         { }
3082         \endpgfscope
3083         \pgfscope
3084         \pgftransformshift
3085         { \pgfpointanchor { \@@_env: - ##1 - #####1 } { south-east } }
3086         \pgfnode
3087         { rectangle }
3088         { center }
3089         {
3090           \hbox
3091             { \pgfsys@markposition { \@@_env: - ##1 - #####1 - SE } }
3092           }
3093         { }
3094         { }
3095         \endpgfscope
3096       }
3097     }
3098   }
3099   \endpgfpicture
3100   \@@_create_extra_nodes:
3101 }
3102 % \end{macrocode}
3103 %
3104 % \bigskip
3105 % The following command creates the diagonal nodes (in fact, if the matrix is
3106 % not a square matrix, not all the nodes are on the diagonal).
3107 % \begin{macrocode}
3108 \cs_new_protected:Npn @@_create_diag_nodes:

```

```

3109 {
3110   \pgfpicture
3111   \pgfrememberpicturepositiononpagetrue
3112   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3113   {
3114     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3115     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3116     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3117     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3118     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3119     \dim_set_eq:NN \l_tmpc_dim \pgf@x
3120     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3121     \dim_set_eq:NN \l_tmpd_dim \pgf@y
3122     \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.

```

3123     \dim_set:Nn \l_tmpa_dim { ( \l_tmpc_dim - \l_tmpa_dim ) / 2 }
3124     \dim_set:Nn \l_tmpb_dim { ( \l_tmpd_dim - \l_tmpb_dim ) / 2 }
3125     \pgfnode { @_diag_node } { center } { } { \@@_env: - ##1 } { }
3126     \str_if_empty:NF \l_@@_name_str
3127     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3128   }

```

Now, the last node. Of course, that is only a coordinate because there is not `.5` anchor for that node.

```

3129   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3130   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3131   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3132   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3133   \pgfcoordinate
3134   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3135   \pgfnodealias
3136   { \@@_env: - last }
3137   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3138   \str_if_empty:NF \l_@@_name_str
3139   {
3140     \pgfnodealias
3141     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3142     { \@@_env: - \int_use:N \l_tmpa_int }
3143     \pgfnodealias
3144     { \l_@@_name_str - last }
3145     { \@@_env: - last }
3146   }
3147   \endpgfpicture
3148 }

```

We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

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

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;

- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
3149 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3150 {
```

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

```
3151 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3152 \int_set:Nn \l_@@_initial_i_int { #1 }
3153 \int_set:Nn \l_@@_initial_j_int { #2 }
3154 \int_set:Nn \l_@@_final_i_int { #1 }
3155 \int_set:Nn \l_@@_final_j_int { #2 }
```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```
3156 \bool_set_false:N \l_@@_stop_loop_bool
3157 \bool_do_until:Nn \l_@@_stop_loop_bool
3158 {
3159   \int_add:Nn \l_@@_final_i_int { #3 }
3160   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3161 \bool_set_false:N \l_@@_final_open_bool
3162 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3163 {
3164   \int_compare:nNnTF { #3 } = 1
3165   { \bool_set_true:N \l_@@_final_open_bool }
3166   {
3167     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3168     { \bool_set_true:N \l_@@_final_open_bool }
3169   }
3170 }
3171 {
3172   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3173   {
3174     \int_compare:nNnTF { #4 } = { -1 }
3175     { \bool_set_true:N \l_@@_final_open_bool }
3176   }
3177   {
3178     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3179     {
3180       \int_compare:nNnTF { #4 } = 1
3181       { \bool_set_true:N \l_@@_final_open_bool }
3182     }
3183   }
3184 }
3185 \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.

```
3186 {
```

We do a step backwards.

```

3187         \int_sub:Nn \l_@@_final_i_int { #3 }
3188         \int_sub:Nn \l_@@_final_j_int { #4 }
3189         \bool_set_true:N \l_@@_stop_loop_bool
3190     }

```

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`.

```

3191     {
3192         \cs_if_exist:cTF
3193         {
3194             @@ _ dotted _
3195             \int_use:N \l_@@_final_i_int -
3196             \int_use:N \l_@@_final_j_int
3197         }
3198         {
3199             \int_sub:Nn \l_@@_final_i_int { #3 }
3200             \int_sub:Nn \l_@@_final_j_int { #4 }
3201             \bool_set_true:N \l_@@_final_open_bool
3202             \bool_set_true:N \l_@@_stop_loop_bool
3203         }
3204         {
3205             \cs_if_exist:cTF
3206             {
3207                 pgf @ sh @ ns @ \@@_env:
3208                 - \int_use:N \l_@@_final_i_int
3209                 - \int_use:N \l_@@_final_j_int
3210             }
3211             { \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.

```

3212         {
3213             \cs_set:cpn
3214             {
3215                 @@ _ dotted _
3216                 \int_use:N \l_@@_final_i_int -
3217                 \int_use:N \l_@@_final_j_int
3218             }
3219             { }
3220         }
3221     }
3222 }
3223 }

```

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

```

3224     \bool_set_false:N \l_@@_stop_loop_bool
3225     \bool_do_until:Nn \l_@@_stop_loop_bool
3226     {
3227         \int_sub:Nn \l_@@_initial_i_int { #3 }
3228         \int_sub:Nn \l_@@_initial_j_int { #4 }
3229         \bool_set_false:N \l_@@_initial_open_bool
3230         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3231         {
3232             \int_compare:nNnTF { #3 } = 1
3233             { \bool_set_true:N \l_@@_initial_open_bool }
3234             {
3235                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }

```

```

3236         { \bool_set_true:N \l_@@_initial_open_bool }
3237     }
3238 }
3239 {
3240     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3241     {
3242         \int_compare:nNnT { #4 } = 1
3243         { \bool_set_true:N \l_@@_initial_open_bool }
3244     }
3245     {
3246         \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3247         {
3248             \int_compare:nNnT { #4 } = { -1 }
3249             { \bool_set_true:N \l_@@_initial_open_bool }
3250         }
3251     }
3252 }
3253 \bool_if:NTF \l_@@_initial_open_bool
3254 {
3255     \int_add:Nn \l_@@_initial_i_int { #3 }
3256     \int_add:Nn \l_@@_initial_j_int { #4 }
3257     \bool_set_true:N \l_@@_stop_loop_bool
3258 }
3259 {
3260     \cs_if_exist:cTF
3261     {
3262         @@ _ dotted _
3263         \int_use:N \l_@@_initial_i_int -
3264         \int_use:N \l_@@_initial_j_int
3265     }
3266     {
3267         \int_add:Nn \l_@@_initial_i_int { #3 }
3268         \int_add:Nn \l_@@_initial_j_int { #4 }
3269         \bool_set_true:N \l_@@_initial_open_bool
3270         \bool_set_true:N \l_@@_stop_loop_bool
3271     }
3272     {
3273         \cs_if_exist:cTF
3274         {
3275             pgf @ sh @ ns @ \@@_env:
3276             - \int_use:N \l_@@_initial_i_int
3277             - \int_use:N \l_@@_initial_j_int
3278         }
3279         { \bool_set_true:N \l_@@_stop_loop_bool }
3280         {
3281             \cs_set:cpn
3282             {
3283                 @@ _ dotted _
3284                 \int_use:N \l_@@_initial_i_int -
3285                 \int_use:N \l_@@_initial_j_int
3286             }
3287             { }
3288         }
3289     }
3290 }
3291 }

```

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.

```

3292 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3293 {
3294     { \int_use:N \l_@@_initial_i_int }
3295     { \int_use:N \l_@@_initial_j_int }
3296     { \int_use:N \l_@@_final_i_int }

```

```

3297         { \int_use:N \l_@@_final_j_int }
3298     }
3299 }

```

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).

```

3300 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3301 {
3302     \int_set:Nn \l_@@_row_min_int 1
3303     \int_set:Nn \l_@@_col_min_int 1
3304     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3305     \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`.

```

3306     \seq_map_inline:Nn \g_@@_submatrix_seq
3307     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
3308 }

```

#1 and #2 are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. #3, #4, #5 and #6 are the specification (in *i* and *j*) of the submatrix where are analysing.

```

3309 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3310 {
3311     \bool_if:nT
3312     {
3313         \int_compare_p:n { #3 <= #1 }
3314         && \int_compare_p:n { #1 <= #5 }
3315         && \int_compare_p:n { #4 <= #2 }
3316         && \int_compare_p:n { #2 <= #6 }
3317     }
3318     {
3319         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3320         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3321         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3322         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3323     }
3324 }

```

```

3325 \cs_new_protected:Npn \@@_set_initial_coords:
3326 {
3327     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3328     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3329 }
3330 \cs_new_protected:Npn \@@_set_final_coords:
3331 {
3332     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3333     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3334 }
3335 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3336 {
3337     \pgfpointanchor
3338     {
3339         \@@_env:
3340         - \int_use:N \l_@@_initial_i_int
3341         - \int_use:N \l_@@_initial_j_int
3342     }
3343     { #1 }
3344     \@@_set_initial_coords:
3345 }
3346 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1

```

```

3347 {
3348   \pgfpointanchor
3349   {
3350     \@@_env:
3351     - \int_use:N \l_@@_final_i_int
3352     - \int_use:N \l_@@_final_j_int
3353   }
3354   { #1 }
3355   \@@_set_final_coords:
3356 }
3357 \cs_new_protected:Npn \@@_open_x_initial_dim:
3358 {
3359   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3360   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3361   {
3362     \cs_if_exist:cT
3363     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3364     {
3365       \pgfpointanchor
3366       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3367       { west }
3368       \dim_set:Nn \l_@@_x_initial_dim
3369       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3370     }
3371   }

```

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

```

3372   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3373   {
3374     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3375     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3376     \dim_add:Nn \l_@@_x_initial_dim \col@sep
3377   }
3378 }
3379 \cs_new_protected:Npn \@@_open_x_final_dim:
3380 {
3381   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3382   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3383   {
3384     \cs_if_exist:cT
3385     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3386     {
3387       \pgfpointanchor
3388       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3389       { east }
3390       \dim_set:Nn \l_@@_x_final_dim
3391       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3392     }
3393   }

```

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

```

3394   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3395   {
3396     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3397     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3398     \dim_sub:Nn \l_@@_x_final_dim \col@sep
3399   }
3400 }

```

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.

```

3401 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3

```

```

3402 {
3403   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3404   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3405   {
3406     \@@_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.

```

3407     \group_begin:
3408     \int_compare:nNnTF { #1 } = 0
3409     { \color { nicematrix-first-row } }
3410     {

```

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.

```

3411         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3412         { \color { nicematrix-last-row } }
3413     }
3414     \keys_set:nn { NiceMatrix / xdots } { #3 }
3415     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3416     \@@_actually_draw_Ldots:
3417   \group_end:
3418 }
3419 }

```

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`.

```

3420 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3421 {
3422   \bool_if:NTF \l_@@_initial_open_bool
3423   {
3424     \@@_open_x_initial_dim:
3425     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3426     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3427   }
3428   { \@@_set_initial_coords_from_anchor:n { base-east } }
3429   \bool_if:NTF \l_@@_final_open_bool
3430   {
3431     \@@_open_x_final_dim:
3432     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3433     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3434   }
3435   { \@@_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 (?).

```

3436   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3437   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3438   \@@_draw_line:
3439 }

```

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.

```

3440 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3441 {
3442   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3443   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3444   {
3445     \@@_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.

```

3446   \group_begin:
3447   \int_compare:nNnTF { #1 } = 0
3448     { \color { nicematrix-first-row } }
3449   {

```

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.

```

3450     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3451     { \color { nicematrix-last-row } }
3452   }
3453   \keys_set:nn { NiceMatrix / xdots } { #3 }
3454   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3455   \@@_actually_draw_Cdots:
3456   \group_end:
3457 }
3458 }

```

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`.

```

3459 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3460 {
3461   \bool_if:NTF \l_@@_initial_open_bool
3462     { \@@_open_x_initial_dim: }
3463     { \@@_set_initial_coords_from_anchor:n { mid-east } }
3464   \bool_if:NTF \l_@@_final_open_bool
3465     { \@@_open_x_final_dim: }
3466     { \@@_set_final_coords_from_anchor:n { mid-west } }
3467   \bool_lazy_and:nnTF
3468     \l_@@_initial_open_bool
3469     \l_@@_final_open_bool
3470   {
3471     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3472     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3473     \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
3474     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3475     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3476   }
3477   {
3478     \bool_if:NT \l_@@_initial_open_bool
3479     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3480     \bool_if:NT \l_@@_final_open_bool
3481     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3482   }

```

```

3483 \@@_draw_line:
3484 }
3485 \cs_new_protected:Npn \@@_open_y_initial_dim:
3486 {
3487   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3488   \dim_set:Nn \l_@@_y_initial_dim
3489     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3490   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3491     {
3492       \cs_if_exist:cT
3493         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3494         {
3495           \pgfpointanchor
3496             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3497             { north }
3498           \dim_set:Nn \l_@@_y_initial_dim
3499             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3500         }
3501     }
3502 }
3503 \cs_new_protected:Npn \@@_open_y_final_dim:
3504 {
3505   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3506   \dim_set:Nn \l_@@_y_final_dim
3507     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3508   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3509     {
3510       \cs_if_exist:cT
3511         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3512         {
3513           \pgfpointanchor
3514             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3515             { south }
3516           \dim_set:Nn \l_@@_y_final_dim
3517             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3518         }
3519     }
3520 }

```

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.

```

3521 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3522 {
3523   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3524   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3525   {
3526     \@@_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.

```

3527   \group_begin:
3528     \int_compare:nNnTF { #2 } = 0
3529     { \color { nicematrix-first-col } }
3530     {
3531       \int_compare:nNnT { #2 } = \l_@@_last_col_int
3532       { \color { nicematrix-last-col } }
3533     }
3534     \keys_set:nn { NiceMatrix / xdots } { #3 }
3535     \tl_if_empty:VF \l_@@_xdots_color_tl
3536     { \color { \l_@@_xdots_color_tl } }
3537     \@@_actually_draw_Vdots:
3538   \group_end:
3539 }
3540 }

```


The command `\l_@@_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`.

```
3541 \cs_new_protected:Npn \l_@@_actually_draw_Vdots:
3542 {
```

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

```
3543   \bool_set_false:N \l_tmpa_bool
```

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

```
3544   \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3545   {
3546     \@@_set_initial_coords_from_anchor:n { south-west }
3547     \@@_set_final_coords_from_anchor:n { north-west }
3548     \bool_set:Nn \l_tmpa_bool
3549     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3550   }
```

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

```
3551   \bool_if:NTF \l_@@_initial_open_bool
3552     \@@_open_y_initial_dim:
3553     { \@@_set_initial_coords_from_anchor:n { south } }
3554   \bool_if:NTF \l_@@_final_open_bool
3555     \@@_open_y_final_dim:
3556     { \@@_set_final_coords_from_anchor:n { north } }
3557   \bool_if:NTF \l_@@_initial_open_bool
3558   {
3559     \bool_if:NTF \l_@@_final_open_bool
3560     {
3561       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3562       \dim_set_eq:NN \l_tmpa_dim \pgf@x
3563       \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3564       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3565       \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3566     }
```

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.

```
3566       \int_compare:nNnT \l_@@_last_col_int > { -2 }
3567       {
3568         \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3569         {
3570           \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3571           \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3572           \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3573           \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3574         }
3575       }
3576     }
3577     { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3578   }
3579   {
3580     \bool_if:NTF \l_@@_final_open_bool
3581     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3582     {
```

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.

```

3583         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3584         {
3585             \dim_set:Nn \l_@@_x_initial_dim
3586             {
3587                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3588                 \l_@@_x_initial_dim \l_@@_x_final_dim
3589             }
3590             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3591         }
3592     }
3593 }
3594 \@@_draw_line:
3595 }

```

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.

```

3596 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3597 {
3598     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3599     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3600     {
3601         \@@_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.

```

3602         \group_begin:
3603         \keys_set:nn { NiceMatrix / xdots } { #3 }
3604         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3605         \@@_actually_draw_Ddots:
3606         \group_end:
3607     }
3608 }

```

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.`

```

3609 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3610 {
3611     \bool_if:NTF \l_@@_initial_open_bool
3612     {
3613         \@@_open_y_initial_dim:
3614         % \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3615         % \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3616         \@@_open_x_initial_dim:
3617     }
3618     { \@@_set_initial_coords_from_anchor:n { south-east } }
3619     \bool_if:NTF \l_@@_final_open_bool

```

```

3620 {
3621   % \@@_open_y_final_dim:
3622   % \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3623   \@@_open_x_final_dim:
3624   \dim_set_eq:Nn \l_@@_x_final_dim \pgf@x
3625 }
3626 { \@@_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.

```

3627   \bool_if:NT \l_@@_parallelize_diags_bool
3628   {
3629     \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).

```

3630     \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.

```

3631     {
3632       \dim_gset:Nn \g_@@_delta_x_one_dim
3633       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3634       \dim_gset:Nn \g_@@_delta_y_one_dim
3635       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3636     }

```

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`.

```

3637     {
3638       \dim_set:Nn \l_@@_y_final_dim
3639       {
3640         \l_@@_y_initial_dim +
3641         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3642         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3643       }
3644     }
3645   }
3646   \@@_draw_line:
3647 }

```

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.

```

3648 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3649 {
3650   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3651   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3652   {
3653     \@@_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.

```

3654     \group_begin:
3655     \keys_set:nn { NiceMatrix / xdots } { #3 }
3656     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3657     \@@_actually_draw_Iddots:
3658   \group_end:
3659 }
3660 }

```

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.

```

3661 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3662 {
3663   \bool_if:NTF \l_@@_initial_open_bool
3664   {
3665     \@@_open_y_initial_dim:
3666     \@@_open_x_initial_dim:
3667   }
3668   { \@@_set_initial_coords_from_anchor:n { south-west } }
3669   \bool_if:NTF \l_@@_final_open_bool
3670   {
3671     \@@_open_y_final_dim:
3672     \@@_open_x_final_dim:
3673   }
3674   { \@@_set_final_coords_from_anchor:n { north-east } }
3675   \bool_if:NT \l_@@_parallelize_diags_bool
3676   {
3677     \int_gincr:N \g_@@_iddots_int
3678     \int_compare:nNnTF \g_@@_iddots_int = 1
3679     {
3680       \dim_gset:Nn \g_@@_delta_x_two_dim
3681       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3682       \dim_gset:Nn \g_@@_delta_y_two_dim
3683       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3684     }
3685     {
3686       \dim_set:Nn \l_@@_y_final_dim
3687       {
3688         \l_@@_y_initial_dim +
3689         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3690         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3691       }
3692     }
3693   }
3694   \@@_draw_line:
3695 }

```

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

```

3696 \cs_new_protected:Npn \@@_draw_line:
3697 {
3698   \pgfrememberpicturepositiononpagetrue
3699   \pgf@relevantforpicturesizefalse
3700   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
3701     \@@_draw_standard_dotted_line:
3702     \@@_draw_non_standard_dotted_line:
3703 }

```

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.

```

3704 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
3705 {
3706   \begin { scope }
3707   \exp_args:No \@@_draw_non_standard_dotted_line:n
3708     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3709 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_non_standard_dotted_line:n` is, in fact, the list of options.

```

3710 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
3711 {
3712   \@@_draw_non_standard_dotted_line:nVV
3713     { #1 }
3714     \l_@@_xdots_up_tl
3715     \l_@@_xdots_down_tl
3716 }
3717 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:nnn #1 #2 #3
3718 {
3719   \draw
3720   [
3721     #1 ,
3722     shorten-> = \l_@@_xdots_shorten_dim ,
3723     shorten-< = \l_@@_xdots_shorten_dim ,
3724   ]
3725     ( \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).

```

3726   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3727   node [ sloped , below ] { $ \scriptstyle #3 $ }
3728   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3729 \end { scope }
3730 }
3731 \cs_generate_variant:Nn \@@_draw_non_standard_dotted_line:nnn { n V V }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real round dots).

```

3732 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3733 {
3734   \bool_lazy_and:nnF
3735     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3736     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3737   {
3738     \pgfscope
3739     \pgftransformshift
3740     {
3741       \pgfpointlineattime { 0.5 }
3742       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3743       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3744     }

```

```

3745 \pgftransformrotate
3746 {
3747   \fp_eval:n
3748   {
3749     atand
3750     (
3751       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3752       \l_@@_x_final_dim - \l_@@_x_initial_dim
3753     )
3754   }
3755 }
3756 \pgfnode
3757 { rectangle }
3758 { south }
3759 {
3760   \c_math_toggle_token
3761   \scriptstyle \l_@@_xdots_up_tl
3762   \c_math_toggle_token
3763 }
3764 { }
3765 { \pgfusepath { } }
3766 \pgfnode
3767 { rectangle }
3768 { north }
3769 {
3770   \c_math_toggle_token
3771   \scriptstyle \l_@@_xdots_down_tl
3772   \c_math_toggle_token
3773 }
3774 { }
3775 { \pgfusepath { } }
3776 \endpgfscope
3777 }
3778 \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.

```

3779 \dim_zero_new:N \l_@@_l_dim
3780 \dim_set:Nn \l_@@_l_dim
3781 {
3782   \fp_to_dim:n
3783   {
3784     sqrt
3785     (
3786       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3787       +
3788       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3789     )
3790   }
3791 }

```

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.

```

3792 \bool_lazy_or:nnF
3793 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3794 { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3795 \@@_draw_standard_dotted_line_i:
3796 \group_end:
3797 }
3798 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3799 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3800 {

```

The number of dots will be $\backslash l_tmpa_int + 1$.

```

3801   \bool_if:NTF \l_@@_initial_open_bool
3802   {
3803     \bool_if:NTF \l_@@_final_open_bool
3804     {
3805       \int_set:Nn \l_tmpa_int
3806       { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3807     }
3808     {
3809       \int_set:Nn \l_tmpa_int
3810       {
3811         \dim_ratio:nn
3812         { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3813         \l_@@_inter_dots_dim
3814       }
3815     }
3816   }
3817   {
3818     \bool_if:NTF \l_@@_final_open_bool
3819     {
3820       \int_set:Nn \l_tmpa_int
3821       {
3822         \dim_ratio:nn
3823         { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3824         \l_@@_inter_dots_dim
3825       }
3826     }
3827     {
3828       \int_set:Nn \l_tmpa_int
3829       {
3830         \dim_ratio:nn
3831         { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3832         \l_@@_inter_dots_dim
3833       }
3834     }
3835   }

```

The dimensions $\backslash l_tmpa_dim$ and $\backslash l_tmpb_dim$ are the coordinates of the vector between two dots in the dotted line.

```

3836   \dim_set:Nn \l_tmpa_dim
3837   {
3838     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3839     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3840   }
3841   \dim_set:Nn \l_tmpb_dim
3842   {
3843     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3844     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3845   }

```

The length ℓ is the length of the dotted line. We note Δ the length between two dots and n the number of intervals between dots. We note $\delta = \frac{1}{2}(\ell - n\Delta)$. The distance between the initial extremity of the line and the first dot will be equal to $k \cdot \delta$ where $k = 0, 1$ or 2 . We first compute this number k in $\backslash l_tmpb_int$.

```

3846   \int_set:Nn \l_tmpb_int
3847   {
3848     \bool_if:NTF \l_@@_initial_open_bool
3849     { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3850     { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3851   }

```

In the loop over the dots, the dimensions $\backslash l_@@_x_initial_dim$ and $\backslash l_@@_y_initial_dim$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

3852 \dim_gadd:Nn \l_@@_x_initial_dim
3853 {
3854   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3855   \dim_ratio:nn
3856   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3857   { 2 \l_@@_l_dim }
3858   * \l_tmpb_int
3859 }
3860 \dim_gadd:Nn \l_@@_y_initial_dim
3861 {
3862   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3863   \dim_ratio:nn
3864   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3865   { 2 \l_@@_l_dim }
3866   * \l_tmpb_int
3867 }
3868 \pgf@relevantforpicturesizefalse
3869 \int_step_inline:nnn 0 \l_tmpa_int
3870 {
3871   \pgfpathcircle
3872   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3873   { \l_@@_radius_dim }
3874   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3875   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3876 }
3877 \pgfusepathqfill
3878 }

```

User commands available in the new environments

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\AtBeginDocument` and the *arg spec* will be rescanned.

```

3879 \AtBeginDocument
3880 {
3881   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3882   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3883   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3884   {
3885     \int_compare:nNnTF \c@jCol = 0
3886     { \@@_error:nn { in~first~col } \Ldots }
3887     {
3888       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3889       { \@@_error:nn { in~last~col } \Ldots }
3890       {
3891         \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
3892         { #1 , down = #2 , up = #3 }
3893       }
3894     }
3895     \bool_if:NF \l_@@_nullify_dots_bool
3896     { \phantom { \ensuremath { \@@_old_ldots } } }
3897     \bool_gset_true:N \g_@@_empty_cell_bool
3898   }

```



```

3899 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3900 {
3901   \int_compare:nNnTF \c@jCol = 0
3902   { \@@_error:nn { in~first~col } \Cdots }
3903   {
3904     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3905     { \@@_error:nn { in~last~col } \Cdots }
3906     {
3907       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
3908       { #1 , down = #2 , up = #3 }
3909     }
3910   }
3911   \bool_if:NF \l_@@_nullify_dots_bool
3912   { \phantom { \ensuremath { \@@_old_cdots } } }
3913   \bool_gset_true:N \g_@@_empty_cell_bool
3914 }

3915 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
3916 {
3917   \int_compare:nNnTF \c@iRow = 0
3918   { \@@_error:nn { in~first~row } \Vdots }
3919   {
3920     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
3921     { \@@_error:nn { in~last~row } \Vdots }
3922     {
3923       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
3924       { #1 , down = #2 , up = #3 }
3925     }
3926   }
3927   \bool_if:NF \l_@@_nullify_dots_bool
3928   { \phantom { \ensuremath { \@@_old_vdots } } }
3929   \bool_gset_true:N \g_@@_empty_cell_bool
3930 }

3931 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
3932 {
3933   \int_case:nnF \c@iRow
3934   {
3935     0 { \@@_error:nn { in~first~row } \Ddots }
3936     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
3937   }
3938   {
3939     \int_case:nnF \c@jCol
3940     {
3941       0 { \@@_error:nn { in~first~col } \Ddots }
3942       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
3943     }
3944     {
3945       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3946       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
3947       { #1 , down = #2 , up = #3 }
3948     }
3949   }
3950 }
3951 \bool_if:NF \l_@@_nullify_dots_bool
3952 { \phantom { \ensuremath { \@@_old_ddots } } }
3953 \bool_gset_true:N \g_@@_empty_cell_bool
3954 }

3955 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
3956 {

```

```

3957 \int_case:nnF \c@iRow
3958 {
3959     0 { \@@_error:nn { in~first~row } \Iddots }
3960     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
3961 }
3962 {
3963     \int_case:nnF \c@jCol
3964     {
3965         0 { \@@_error:nn { in~first~col } \Iddots }
3966         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
3967     }
3968     {
3969         \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3970         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
3971         { #1 , down = #2 , up = #3 }
3972     }
3973 }
3974 \bool_if:NF \l_@@_nullify_dots_bool
3975 { \phantom { \ensuremath { \@@_old_iddots } } }
3976 \bool_gset_true:N \g_@@_empty_cell_bool
3977 }
3978 }

```

End of the `\AtBeginDocument`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

3979 \keys_define:nn { NiceMatrix / Ddots }
3980 {
3981     draw-first .bool_set:N = \l_@@_draw_first_bool ,
3982     draw-first .default:n = true ,
3983     draw-first .value_forbidden:n = true
3984 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

3985 \cs_new_protected:Npn \@@_Hspace:
3986 {
3987     \bool_gset_true:N \g_@@_empty_cell_bool
3988     \hspace
3989 }

```

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`.

```

3990 \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`.

```

3991 \cs_new:Npn \@@_Hdotsfor:
3992 {
3993     \bool_lazy_and:nnTF
3994     { \int_compare_p:nNn \c@jCol = 0 }
3995     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
3996     {
3997         \bool_if:NTF \g_@@_after_col_zero_bool
3998         {
3999             \multicolumn { 1 } { c } { }
4000             \@@_Hdotsfor_i
4001         }
4002         { \@@_fatal:n { Hdotsfor~in~col~0 } }
4003     }
4004 }

```

```

4005     \multicolumn { 1 } { c } { }
4006     \@@_Hdotsfor_i
4007 }
4008 }

```

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:`).

```

4009 \AtBeginDocument
4010 {
4011     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4012     \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.

```

4013     \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4014     {
4015         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4016         {
4017             \@@_Hdotsfor:nnnn
4018             { \int_use:N \c@iRow }
4019             { \int_use:N \c@jCol }
4020             { #2 }
4021             {
4022                 #1 , #3 ,
4023                 down = \exp_not:n { #4 } ,
4024                 up = \exp_not:n { #5 }
4025             }
4026         }
4027         \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4028     }
4029 }

```

Enf of `\AtBeginDocument`.

```

4030 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4031 {
4032     \bool_set_false:N \l_@@_initial_open_bool
4033     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4034     \int_set:Nn \l_@@_initial_i_int { #1 }
4035     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

4036     \int_compare:nNnTF { #2 } = 1
4037     {
4038         \int_set:Nn \l_@@_initial_j_int 1
4039         \bool_set_true:N \l_@@_initial_open_bool
4040     }
4041     {
4042         \cs_if_exist:cTF
4043         {
4044             pgf @ sh @ ns @ \@@_env:
4045             - \int_use:N \l_@@_initial_i_int
4046             - \int_eval:n { #2 - 1 }
4047         }
4048         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4049         {
4050             \int_set:Nn \l_@@_initial_j_int { #2 }
4051             \bool_set_true:N \l_@@_initial_open_bool
4052         }
4053     }
4054     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4055     {

```

```

4056     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4057     \bool_set_true:N \l_@@_final_open_bool
4058   }
4059   {
4060     \cs_if_exist:cTF
4061     {
4062       pgf @ sh @ ns @ \@@_env:
4063       - \int_use:N \l_@@_final_i_int
4064       - \int_eval:n { #2 + #3 }
4065     }
4066     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4067     {
4068       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4069       \bool_set_true:N \l_@@_final_open_bool
4070     }
4071   }
4072   \group_begin:
4073   \int_compare:nNnTF { #1 } = 0
4074   { \color { nicematrix-first-row } }
4075   {
4076     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4077     { \color { nicematrix-last-row } }
4078   }
4079   \keys_set:nn { NiceMatrix / xdots } { #4 }
4080   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4081   \@@_actually_draw_Ldots:
4082   \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).

```

4083   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4084   { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4085 }

4086 \AtBeginDocument
4087 {
4088   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4089   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4090   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4091   {
4092     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4093     {
4094       \@@_Vdotsfor:nnnn
4095       { \int_use:N \c@iRow }
4096       { \int_use:N \c@jCol }
4097       { #2 }
4098       {
4099         #1 , #3 ,
4100         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4101       }
4102     }
4103   }
4104 }

```

Enf of `\AtBeginDocument`.

```

4105 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4106 {
4107   \bool_set_false:N \l_@@_initial_open_bool
4108   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```
4109 \int_set:Nn \l_@@_initial_j_int { #2 }
4110 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

For the row, it's a bit more complicated.

```
4111 \int_compare:nNnTF #1 = 1
4112 {
4113   \int_set:Nn \l_@@_initial_i_int 1
4114   \bool_set_true:N \l_@@_initial_open_bool
4115 }
4116 {
4117   \cs_if_exist:cTF
4118   {
4119     pgf @ sh @ ns @ \@@_env:
4120     - \int_eval:n { #1 - 1 }
4121     - \int_use:N \l_@@_initial_j_int
4122   }
4123   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4124   {
4125     \int_set:Nn \l_@@_initial_i_int { #1 }
4126     \bool_set_true:N \l_@@_initial_open_bool
4127   }
4128 }
4129 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4130 {
4131   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4132   \bool_set_true:N \l_@@_final_open_bool
4133 }
4134 {
4135   \cs_if_exist:cTF
4136   {
4137     pgf @ sh @ ns @ \@@_env:
4138     - \int_eval:n { #1 + #3 }
4139     - \int_use:N \l_@@_final_j_int
4140   }
4141   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4142   {
4143     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4144     \bool_set_true:N \l_@@_final_open_bool
4145   }
4146 }
4147 \group_begin:
4148 \int_compare:nNnTF { #2 } = 0
4149 { \color { nicematrix-first-col } }
4150 {
4151   \int_compare:nNnT { #2 } = \g_@@_col_total_int
4152   { \color { nicematrix-last-col } }
4153 }
4154 \keys_set:nn { NiceMatrix / xdots } { #4 }
4155 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4156 \@@_actually_draw_Vdots:
4157 \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).

```
4158 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4159 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4160 }
```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```
4161 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }
```

The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells.

First, we write a command with an argument of the format i - j and applies the command `\int_eval:n` to i and j ; this must *not* be protected (and is, of course fully expandable).⁶³

```
4162 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4163 { \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).

```
4164 \AtBeginDocument
4165 {
4166   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4167   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4168   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4169     {
4170       \group_begin:
4171       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4172       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4173       \use:e
4174       {
4175         \@@_line_i:nn
4176         { \@@_double_int_eval:n #2 \q_stop }
4177         { \@@_double_int_eval:n #3 \q_stop }
4178       }
4179       \group_end:
4180     }
4181 }

4182 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4183 {
4184   \bool_set_false:N \l_@@_initial_open_bool
4185   \bool_set_false:N \l_@@_final_open_bool
4186   \bool_if:nTF
4187     {
4188       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4189       ||
4190       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4191     }
4192     {
4193       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4194     }
4195     { \@@_draw_line_ii:nn { #1 } { #2 } }
4196 }

4197 \AtBeginDocument
4198 {
4199   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4200   {
```

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:.`

⁶³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

4201     \c_@@_pgfortikzpicture_tl
4202     \@@_draw_line_iii:nn { #1 } { #2 }
4203     \c_@@_endpgfortikzpicture_tl
4204   }
4205 }

```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```

4206 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4207 {
4208   \pgfrememberpicturepositiononpagetrue
4209   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4210   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4211   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4212   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4213   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4214   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4215   \@@_draw_line:
4216 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That's why we try to draw rectangles of the same color in the same instruction `\pgfusepath { fill }` (and they will be in the same instruction `fill`—coded `f`—in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor`, `\@@_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).

```

4217 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4218 {

```

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.

```

4219   \int_zero:N \l_tmpa_int
4220   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4221     { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4222   \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

4223   {
4224     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4225     \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4226   }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

4227     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4228   }

4229 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4230 \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`).

```

4231 \cs_new_protected:Npn \@@_actually_color:
4232 {
4233   \pgfpicture
4234   \pgf@relevantforpicturesizefalse
4235   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4236     {
4237       \color ##2
4238       \use:c { g_@@_color _ ##1 _tl }
4239       \tl_gclear:c { g_@@_color _ ##1 _tl }
4240       \pgfusepath { fill }
4241     }
4242   \endpgfpicture
4243 }

4244 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
4245 {
4246   \tl_set:Nn \l_tmpa_tl { #1 }
4247   \tl_set:Nn \l_tmpb_tl { #2 }
4248 }

4249 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4250 {
4251   \tl_set:Nn \l_@@_rows_tl { #1 }
4252   \tl_set:Nn \l_@@_cols_tl { #2 }
4253   \@@_cartesian_path:
4254 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

4255 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4256 {
4257   \tl_if_blank:nF { #2 }
4258   {
4259     \@@_add_to_colors_seq:xn
4260     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4261     { \@@_cartesian_color:nn { #3 } { - } }
4262   }
4263 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

4264 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4265 {
4266   \tl_if_blank:nF { #2 }
4267   {
4268     \@@_add_to_colors_seq:xn
4269     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4270     { \@@_cartesian_color:nn { - } { #3 } }
4271   }
4272 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

4273 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4274 {

```



```

4275 \tl_if_blank:nF { #2 }
4276 {
4277   \@@_add_to_colors_seq:xn
4278   { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4279   { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4280 }
4281 }

```

The last argument is the radius of the corners of the rectangle.

```

4282 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4283 {
4284   \tl_if_blank:nF { #2 }
4285   {
4286     \@@_add_to_colors_seq:xn
4287     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4288     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4289   }
4290 }

```

The last argument is the radius of the corners of the rectangle.

```

4291 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4292 {
4293   \@@_cut_on_hyphen:w #1 \q_stop
4294   \tl_clear_new:N \l_tmpc_tl
4295   \tl_clear_new:N \l_tmpd_tl
4296   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
4297   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
4298   \@@_cut_on_hyphen:w #2 \q_stop
4299   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
4300   \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`.

```

4301 \@@_cartesian_path:n { #3 }
4302 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

4303 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4304 {
4305   \clist_map_inline:nn { #3 }
4306   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4307 }

```

```

4308 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4309 {
4310   \int_step_inline:nn { \int_use:N \c@iRow }
4311   {
4312     \int_step_inline:nn { \int_use:N \c@jCol }
4313     {
4314       \int_if_even:nTF { ####1 + ##1 }
4315       { \@@_cellcolor [ #1 ] { #2 } }
4316       { \@@_cellcolor [ #1 ] { #3 } }
4317     } { ##1 - ####1 }
4318   }
4319 }
4320 }

```

```

4321 \keys_define:nn { NiceMatrix / arraycolor }
4322 { except-corners .code:n = \@@_error:n { key except-corners } }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns). The third argument is a optional argument which a list of pairs key-value.

```

4323 \NewDocumentCommand \@@_arraycolor { 0 { } m 0 { } }
4324 {
4325   \keys_set:nn { NiceMatrix / arraycolor } { #3 }
4326   \@@_rectanglecolor [ #1 ] { #2 }
4327   { 1 - 1 }
4328   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4329 }

4330 \keys_define:nn { NiceMatrix / rowcolors }
4331 {
4332   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4333   respect-blocks .default:n = true ,
4334   cols .tl_set:N = \l_@@_cols_tl ,
4335   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4336   restart .default:n = true ,
4337   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4338 }

```

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.

```

4339 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
4340 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

4341   \group_begin:
4342   \seq_clear_new:N \l_@@_colors_seq
4343   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
4344   \tl_clear_new:N \l_@@_cols_tl
4345   \tl_set:Nn \l_@@_cols_tl { - }
4346   \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).

```

4347   \int_zero_new:N \l_@@_color_int
4348   \int_set:Nn \l_@@_color_int 1
4349   \bool_if:NT \l_@@_respect_blocks_bool
4350   {

```

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`).

```

4351     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4352     \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
4353     { \@@_not_in_exterior_p:nnnn ##1 }
4354   }
4355   \pgfpicture
4356   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

4357   \clist_map_inline:nn { #2 }
4358   {
4359     \tl_set:Nn \l_tmpa_tl { ##1 }
4360     \tl_if_in:NnTF \l_tmpa_tl { - }
4361     { \@@_cut_on_hyphen:w ##1 \q_stop }
4362     { \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.

```

4363 \int_set:Nn \l_tmpa_int \l_tmpa_tl
4364 \bool_if:NTF \l_@@_rowcolors_restart_bool
4365 { \int_set:Nn \l_@@_color_int 1 }
4366 { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
4367 \int_zero_new:N \l_tmpc_int
4368 \int_set:Nn \l_tmpc_int \l_tmpb_tl
4369 \int_do_until:nNnn \l_tmpa_int > \l_tmpc_int
4370 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4371 \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).

```

4372 \bool_if:NT \l_@@_respect_blocks_bool
4373 {
4374   \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
4375   { \@@_intersect_our_row_p:nnnn #####1 }
4376   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4377 }
4378 \tl_set:Nx \l_@@_rows_tl
4379 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_tmpc_tl` will be the color that we will use.

```

4380 \tl_clear_new:N \l_@@_color_tl
4381 \tl_set:Nx \l_@@_color_tl
4382 {
4383   \@@_color_index:n
4384   {
4385     \int_mod:nn
4386     { \l_@@_color_int - 1 }
4387     { \seq_count:N \l_@@_colors_seq }
4388     + 1
4389   }
4390 }
4391 \tl_if_empty:NF \l_@@_color_tl
4392 {
4393   \@@_add_to_colors_seq:xx
4394   { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
4395   { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
4396 }
4397 \int_incr:N \l_@@_color_int
4398 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4399 }
4400 }
4401 \endpgfpicture
4402 \group_end:
4403 }

```

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.

```

4404 \cs_new:Npn \@@_color_index:n #1
4405 {
4406   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
4407   { \@@_color_index:n { #1 - 1 } }
4408   { \seq_item:Nn \l_@@_colors_seq { #1 } }
4409 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

4410 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4411 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

```

```

4412 \cs_new_protected:Npn \@@_rowcolors_i:nnnn #1 #2 #3 #4
4413 {
4414   \int_compare:nNt { #3 } > \l_tmpb_int
4415   { \int_set:Nn \l_tmpb_int { #3 } }
4416 }

4417 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnn p
4418 {
4419   \bool_lazy_or:nnTF
4420   { \int_compare_p:nNn { #4 } = \c_zero_int }
4421   { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
4422   \prg_return_false:
4423   \prg_return_true:
4424 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

4425 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnn p
4426 {
4427   \bool_if:nTF
4428   {
4429     \int_compare_p:n { #1 <= \l_tmpa_int }
4430     &&
4431     \int_compare_p:n { \l_tmpa_int <= #3 }
4432   }
4433   \prg_return_true:
4434   \prg_return_false:
4435 }

```

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).

```

4436 \cs_new_protected:Npn \@@_cartesian_path:n #1
4437 {
4438   \bool_lazy_and:nnT
4439   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4440   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4441   {
4442     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4443     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4444   }

```

We begin the loop over the columns.

```

4445 \clist_map_inline:Nn \l_@@_cols_tl
4446 {
4447   \tl_set:Nn \l_tmpa_tl { ##1 }
4448   \tl_if_in:NnTF \l_tmpa_tl { - }
4449   { \@@_cut_on_hyphen:w ##1 \q_stop }
4450   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4451   \bool_lazy_or:nnT
4452   { \tl_if_blank_p:V \l_tmpa_tl }
4453   { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4454   { \tl_set:Nn \l_tmpa_tl { 1 } }
4455   \bool_lazy_or:nnT
4456   { \tl_if_blank_p:V \l_tmpb_tl }
4457   { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4458   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4459   \int_compare:nNt \l_tmpb_tl > \c@jCol
4460   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

\l_tmpc_tl will contain the number of column.

```
4461 \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.

```
4462 \@@_qpoint:n { col - \l_tmpa_tl }
4463 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4464 { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4465 { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4466 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4467 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
4468 \clist_map_inline:Nn \l_@@_rows_tl
4469 {
4470 \tl_set:Nn \l_tmpa_tl { #####1 }
4471 \tl_if_in:NnTF \l_tmpa_tl { - }
4472 { \@@_cut_on_hyphen:w #####1 \q_stop }
4473 { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4474 \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4475 \tl_if_empty:NT \l_tmpb_tl
4476 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4477 \int_compare:nNnT \l_tmpb_tl > \c@iRow
4478 { \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.

```
4479 \seq_if_in:NxF \l_@@_corners_cells_seq
4480 { \l_tmpa_tl - \l_tmpc_tl }
4481 {
4482 \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
4483 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4484 \@@_qpoint:n { row - \l_tmpa_tl }
4485 \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4486 \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4487 \pgfpathrectanglecorners
4488 { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4489 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4490 }
4491 }
4492 }
4493 }
```

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).

```
4494 \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.

```
4495 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4496 {
4497 \clist_set_eq:NN \l_tmpa_clist #1
4498 \clist_clear:N #1
4499 \clist_map_inline:Nn \l_tmpa_clist
4500 {
4501 \tl_set:Nn \l_tmpa_tl { ##1 }
4502 \tl_if_in:NnTF \l_tmpa_tl { - }
4503 { \@@_cut_on_hyphen:w ##1 \q_stop }
4504 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4505 \bool_lazy_or:nnT
4506 { \tl_if_blank_p:V \l_tmpa_tl }
4507 { \str_if_eq_p:Vn \l_tmpa_tl { * } }
```

```

4508     { \tl_set:Nn \l_tmpa_tl { 1 } }
4509   \bool_lazy_or:nnT
4510     { \tl_if_blank_p:V \l_tmpb_tl }
4511     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4512     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4513   \int_compare:nNnT \l_tmpb_tl > #2
4514     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4515   \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4516     { \clist_put_right:Nn #1 { ####1 } }
4517 }
4518 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the `tabular`.

```

4519 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4520 {
4521   \peek_remove_spaces:n
4522   {
4523     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4524     {

```

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`).

```

4525       \cellcolor [ #1 ] { \exp_not:n { #2 } }
4526       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4527     }
4528   }
4529 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the `tabular`.

```

4530 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4531 {
4532   \peek_remove_spaces:n
4533   {
4534     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4535     {
4536       \exp_not:N \rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4537       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4538       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4539     }
4540   }
4541 }

```

```

4542 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4543 {

```

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).

```

4544   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4545   {

```

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).

```

4546     \tl_gput_left:Nx \g_nicematrix_code_before_tl
4547     {
4548       \exp_not:N \columncolor [ #1 ]
4549       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4550     }
4551   }
4552 }

```

The vertical rules

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
4553 \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`.

```
4554 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4555 {
4556   \int_compare:nNnTF \l_@@_first_col_int = 0
4557     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4558     {
4559       \int_compare:nNnTF \c@jCol = 0
4560       {
4561         \int_compare:nNnF \c@iRow = { -1 }
4562         { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4563       }
4564       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4565     }
4566 }
```

This definition may seem complicated by we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell.

The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```
4567 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4568 {
4569   \int_compare:nNnF \c@iRow = 0
4570   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4571 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the column `#1` (that is to say on the left side). `#2` is the number of consecutive occurrences of `|`.

```
4572 \cs_new_protected:Npn \@@_vline:nn #1 #2
4573 {
```

The following test is for the case where the user don't use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```
4574   \int_compare:nNnT { #1 } < { \c@jCol + 2 }
4575   {
4576     \pgfpicture
4577     \@@_vline_i:nn { #1 } { #2 }
4578     \endpgfpicture
4579   }
```

```
4581 \cs_new_protected:Npn \@@_vline_i:nn #1 #2
4582 {
```

`\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`.

```
4583   \tl_set:Nx \l_tmpb_tl { #1 }
4584   \tl_clear_new:N \l_tmpc_tl
4585   \int_step_variable:nNn \c@iRow \l_tmpa_tl
4586   {
```

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.

```

4587     \bool_gset_true:N \g_tmpa_bool
4588     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4589       { \@@_test_vline_in_block:nnnn ##1 }
4590     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4591       { \@@_test_vline_in_block:nnnn ##1 }
4592     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4593       { \@@_test_vline_in_stroken_block:nnnn ##1 }
4594     \clist_if_empty:NF \l_@@_corners_clist
4595     \@@_test_in_corner_v:
4596     \bool_if:NTF \g_tmpa_bool
4597     {
4598       \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4599       { \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl }
4600     }
4601     {
4602       \tl_if_empty:NF \l_tmpc_tl
4603       {
4604         \@@_vline_ii:nnnn
4605         { #1 }
4606         { #2 }
4607         \l_tmpc_tl
4608         { \int_eval:n { \l_tmpa_tl - 1 } }
4609         \tl_clear:N \l_tmpc_tl
4610       }
4611     }
4612   }
4613   \tl_if_empty:NF \l_tmpc_tl
4614   {
4615     \@@_vline_ii:nnnn
4616     { #1 }
4617     { #2 }
4618     \l_tmpc_tl
4619     { \int_use:N \c@iRow }
4620     \tl_clear:N \l_tmpc_tl
4621   }
4622 }

```

```

4623 \cs_new_protected:Npn \@@_test_in_corner_v:
4624 {
4625   \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
4626   {
4627     \seq_if_in:NxT
4628       \l_@@_corners_cells_seq
4629       { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4630       { \bool_set_false:N \g_tmpa_bool }
4631   }
4632   {
4633     \seq_if_in:NxT
4634       \l_@@_corners_cells_seq
4635       { \l_tmpa_tl - \l_tmpb_tl }
4636     {
4637       \int_compare:nNnTF \l_tmpb_tl = 1
4638       { \bool_set_false:N \g_tmpa_bool }
4639     }
4640     \seq_if_in:NxT
4641       \l_@@_corners_cells_seq
4642       { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }

```



```

4643         { \bool_set_false:N \g_tmpa_bool }
4644     }
4645 }
4646 }
4647 }

```

#1 is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the numbers of the rows between which the rule has to be drawn.

```

4648 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
4649 {
4650     \pgfrememberpicturepositiononpagetrue
4651     \pgf@relevantforpicturesizefalse
4652     \@@_qpoint:n { row - #3 }
4653     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4654     \@@_qpoint:n { col - #1 }
4655     \dim_set_eq:NN \l_tmpb_dim \pgf@x
4656     \@@_qpoint:n { row - \@@_succ:n { #4 } }
4657     \dim_set_eq:NN \l_tmpc_dim \pgf@y
4658     \bool_lazy_all:nT
4659     {
4660         { \int_compare_p:nNn { #2 } > 1 }
4661         { \cs_if_exist_p:N \CT@drsc@ } % condition added in version 5.18a
4662         { ! \tl_if_blank_p:V \CT@drsc@ }
4663     }
4664     {
4665         \group_begin:
4666         \CT@drsc@
4667         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4668         \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
4669         \dim_set:Nn \l_tmpd_dim
4670             { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4671         \pgfpathrectanglecorners
4672             { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4673             { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4674         \pgfusepath { fill }
4675         \group_end:
4676     }
4677     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4678     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4679     \prg_replicate:nn { #2 - 1 }
4680     {
4681         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4682         \dim_sub:Nn \l_tmpb_dim \doublerulesep
4683         \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4684         \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4685     }
4686     \CT@arc@
4687     \pgfsetlinewidth { 1.1 \arrayrulewidth }
4688     \pgfsetrectcap
4689     \pgfusepathqstroke
4690 }

```

The following command draws a complete vertical rule in the column #1 (#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key `corners` is not used).

```

4691 \cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
4692 { \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

```

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).

```

4693 \cs_new_protected:Npn \@@_draw_vlines:

```

```

4694 {
4695   \int_step_inline:nnn
4696   {
4697     \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4698     1 2
4699   }
4700   {
4701     \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4702     { \@@_succ:n \c@jCol }
4703     \c@jCol
4704   }
4705   {
4706     \tl_if_eq:NnF \l_@@_vlines_clist { all }
4707     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4708     { \@@_vline:nn { ##1 } 1 }
4709   }
4710 }

```

The horizontal rules

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the row #1. #2 is the number of consecutive occurrences of `\Hline`.

```

4711 \cs_new_protected:Npn \@@_hline:nn #1 #2
4712 {
4713   \pgfpicture
4714   \@@_hline_i:nn { #1 } { #2 }
4715   \endpgfpicture
4716 }
4717 \cs_new_protected:Npn \@@_hline_i:nn #1 #2
4718 {

```

`\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`.

```

4719   \tl_set:Nn \l_tmpa_tl { #1 }
4720   \tl_clear_new:N \l_tmpc_tl
4721   \int_step_variable:nNn \c@jCol \l_tmpb_tl
4722   {

```

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.

```

4723     \bool_gset_true:N \g_tmpa_bool
4724     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4725     { \@@_test_hline_in_block:nnnn ##1 }
4726     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4727     { \@@_test_hline_in_block:nnnn ##1 }
4728     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4729     { \@@_test_hline_in_stroken_block:nnnn ##1 }
4730     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4731     \bool_if:NTF \g_tmpa_bool
4732     {
4733       \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4734       { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
4735     }
4736   {
4737     \tl_if_empty:NF \l_tmpc_tl
4738     {
4739       \@@_hline_ii:nnnn

```

```

4740         { #1 }
4741         { #2 }
4742         \l_tmpc_tl
4743         { \int_eval:n { \l_tmpb_tl - 1 } }
4744         \tl_clear:N \l_tmpc_tl
4745     }
4746 }
4747 }
4748 \tl_if_empty:NF \l_tmpc_tl
4749 {
4750     \@@_hline_ii:nnnn
4751     { #1 }
4752     { #2 }
4753     \l_tmpc_tl
4754     { \int_use:N \c_jCol }
4755     \tl_clear:N \l_tmpc_tl
4756 }
4757 }

4758 \cs_new_protected:Npn \@@_test_in_corner_h:
4759 {
4760     \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c_iRow }
4761     {
4762         \seq_if_in:NxT
4763         \l_@@_corners_cells_seq
4764         { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4765         { \bool_set_false:N \g_tmpa_bool }
4766     }
4767     {
4768         \seq_if_in:NxT
4769         \l_@@_corners_cells_seq
4770         { \l_tmpa_tl - \l_tmpb_tl }
4771         {
4772             \int_compare:nNnTF \l_tmpa_tl = 1
4773             { \bool_set_false:N \g_tmpa_bool }
4774             {
4775                 \seq_if_in:NxT
4776                 \l_@@_corners_cells_seq
4777                 { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4778                 { \bool_set_false:N \g_tmpa_bool }
4779             }
4780         }
4781     }
4782 }

```

#1 is the number of the row; #2 is the number of horizontal rules to draw (with potentially a color between); #3 and #4 are the number of the columns between which the rule has to be drawn.

```

4783 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
4784 {
4785     \pgfrememberpicturepositiononpagetrue
4786     \pgf@relevantforpicturesizefalse
4787     \@@_qpoint:n { col - #3 }
4788     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4789     \@@_qpoint:n { row - #1 }
4790     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4791     \@@_qpoint:n { col - \@@_succ:n { #4 } }
4792     \dim_set_eq:NN \l_tmpc_dim \pgf@x
4793     \bool_lazy_all:nT
4794     {
4795         { \int_compare_p:nNn { #2 } > 1 }
4796         { \cs_if_exist_p:N \CT@drsc@ } % condition added in version 6.0
4797         { ! \tl_if_blank_p:V \CT@drsc@ }

```

```

4798     }
4799     {
4800         \group_begin:
4801         \CT@drsc@
4802         \dim_set:Nn \l_tmpd_dim
4803             { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4804         \pgfpathrectanglecorners
4805             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4806             { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4807         \pgfusepathqfill
4808         \group_end:
4809     }
4810     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4811     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4812     \prg_replicate:nn { #2 - 1 }
4813     {
4814         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4815         \dim_sub:Nn \l_tmpb_dim \doublerulesep
4816         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4817         \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4818     }
4819     \CT@arc@
4820     \pgfsetlinewidth { 1.1 \arrayrulewidth }
4821     \pgfsetrectcap
4822     \pgfusepathqstroke
4823 }

4824 \cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
4825 { \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as `\Cdots` and in the corners (if the key `corners` is used)).

```

4826 \cs_new_protected:Npn \@@_draw_hlines:
4827 {
4828     \int_step_inline:nnn
4829     {
4830         \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4831             1 2
4832     }
4833     {
4834         \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4835         { \@@_succ:n \c@iRow }
4836         \c@iRow
4837     }
4838     {
4839         \tl_if_eq:NnF \l_@@_hlines_clist { all }
4840         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
4841         { \@@_hline:nn { ##1 } 1 }
4842     }
4843 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

4844 \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.

```

4845 \cs_set:Npn \@@_Hline_i:n #1
4846 {
4847     \peek_meaning_ignore_spaces:NTF \Hline
4848     { \@@_Hline_ii:nn { #1 + 1 } }
4849     { \@@_Hline_iii:n { #1 } }
4850 }

```

```

4851 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
4852 \cs_set:Npn \@@_Hline_iii:n #1
4853 {
4854   \skip_vertical:n
4855   {
4856     \arrayrulewidth * ( #1 )
4857     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
4858   }
4859   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4860   { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
4861   \ifnum 0 = `{ \fi }
4862 }

```

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`.

```

4863 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4
4864 {
4865   \bool_lazy_all:nT
4866   {
4867     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
4868     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4869     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4870     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4871   }
4872   { \bool_gset_false:N \g_tmpa_bool }
4873 }

```

The same for vertical rules.

```

4874 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4
4875 {
4876   \bool_lazy_all:nT
4877   {
4878     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4879     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4880     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
4881     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4882   }
4883   { \bool_gset_false:N \g_tmpa_bool }
4884 }
4885 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
4886 {
4887   \bool_lazy_all:nT
4888   {
4889     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4890     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
4891     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4892     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4893   }
4894   { \bool_gset_false:N \g_tmpa_bool }
4895 }
4896 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
4897 {
4898   \bool_lazy_all:nT
4899   {
4900     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4901     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4902     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }

```

```

4903     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
4904   }
4905   { \bool_gset_false:N \g_tmpa_bool }
4906 }

```

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.

```

4907 \cs_new_protected:Npn \@@_compute_corners:
4908 {

```

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.

```

4909   \seq_clear_new:N \l_@@_corners_cells_seq
4910   \clist_map_inline:Nn \l_@@_corners_clist
4911   {
4912     \str_case:nnF { ##1 }
4913     {
4914       { NW }
4915       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
4916       { NE }
4917       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
4918       { SW }
4919       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
4920       { SE }
4921       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
4922     }
4923     { \@@_error:nn { bad~corner } { ##1 } }
4924   }

```

Even if the user has used the key `corners` (or the key `hvlines-except-corners`), the list of cells in the corners may be empty.

```

4925   \seq_if_empty:NF \l_@@_corners_cells_seq
4926   {

```

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.

```

4927     \tl_gput_right:Nx \g_@@_aux_tl
4928     {
4929       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
4930       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
4931     }
4932   }
4933 }

```

“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.

```

4934 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
4935 {

```

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.

```

4936 \bool_set_false:N \l_tmpa_bool
4937 \int_zero_new:N \l_@@_last_empty_row_int
4938 \int_set:Nn \l_@@_last_empty_row_int { #1 }
4939 \int_step_inline:nnnn { #1 } { #3 } { #5 }
4940 {
4941   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
4942   \bool_lazy_or:nnTF
4943   {
4944     \cs_if_exist_p:c
4945     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
4946   }
4947   \l_tmpb_bool
4948   { \bool_set_true:N \l_tmpa_bool }
4949   {
4950     \bool_if:NF \l_tmpa_bool
4951     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
4952   }
4953 }

```

Now, you determine the last empty cell in the row of number 1.

```

4954 \bool_set_false:N \l_tmpa_bool
4955 \int_zero_new:N \l_@@_last_empty_column_int
4956 \int_set:Nn \l_@@_last_empty_column_int { #2 }
4957 \int_step_inline:nnnn { #2 } { #4 } { #6 }
4958 {
4959   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
4960   \bool_lazy_or:nnTF
4961   \l_tmpb_bool
4962   {
4963     \cs_if_exist_p:c
4964     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
4965   }
4966   { \bool_set_true:N \l_tmpa_bool }
4967   {
4968     \bool_if:NF \l_tmpa_bool
4969     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
4970   }
4971 }

```

Now, we loop over the rows.

```

4972 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
4973 {

```

We treat the row number `##1` with another loop.

```

4974   \bool_set_false:N \l_tmpa_bool
4975   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
4976   {
4977     \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
4978     \bool_lazy_or:nnTF
4979     \l_tmpb_bool
4980     {
4981       \cs_if_exist_p:c
4982       { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
4983     }
4984     { \bool_set_true:N \l_tmpa_bool }
4985     {
4986       \bool_if:NF \l_tmpa_bool
4987       {
4988         \int_set:Nn \l_@@_last_empty_column_int { ####1 }

```

```

4989             \seq_put_right:Nn
4990             \l_@@_corners_cells_seq
4991             { ##1 - #####1 }
4992         }
4993     }
4994 }
4995 }
4996 }

```

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`).

```

4997 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
4998 {
4999     \int_set:Nn \l_tmpa_int { #1 }
5000     \int_set:Nn \l_tmpb_int { #2 }
5001     \bool_set_false:N \l_tmpb_bool
5002     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5003     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
5004 }
5005 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6
5006 {
5007     \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
5008     {
5009         \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
5010         {
5011             \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
5012             {
5013                 \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
5014                 { \bool_set_true:N \l_tmpb_bool }
5015             }
5016         }
5017     }
5018 }

```

The commands to draw dotted lines to separate columns and rows

These commands don't use the normal nodes, the medium nor the large nodes. They only use the `col` nodes and the `row` nodes.

Horizontal dotted lines

The following command must *not* be protected because it's meant to be expanded in a `\noalign`.

```

5019 \cs_new:Npn \@@_hdottedline:
5020 {
5021     \noalign { \skip_vertical:N 2\l_@@_radius_dim }
5022     \@@_hdottedline_i:
5023 }

```

On the other side, the following command should be protected.

```

5024 \cs_new_protected:Npn \@@_hdottedline_i:
5025 {

```

We write in the code-after the instruction that will actually draw the dotted line. It's not possible to draw this dotted line now because we don't know the length of the line (we don't even know the number of columns).

```

5026     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5027     { \@@_hdottedline:n { \int_use:N \c@iRow } }
5028 }

```


The command `\@@_hdottedline:n` is the command written in the `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```
5029 \AtBeginDocument
5030 {
```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That’s why we construct now a version of `\@@_hdottedline:n` with the right environment (`\begin{pgfpicture}\end{pgfpicture}` or `\begin{tikzpicture}...\end{tikzpicture}`).

```
5031 \cs_new_protected:Npx \@@_hdottedline:n #1
5032 {
5033   \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
5034   \bool_set_true:N \exp_not:N \l_@@_final_open_bool
5035   \c_@@_pgfortikzpicture_tl
5036   \@@_hdottedline_i:n { #1 }
5037   \c_@@_endpgfortikzpicture_tl
5038 }
5039 }
```

The following command *must* be protected since it is used in the construction of `\@@_hdottedline:n`.

```
5040 \cs_new_protected:Npn \@@_hdottedline_i:n #1
5041 {
5042   \pgfrememberpicturepositiononpagetrue
5043   \@@_qpoint:n { row - #1 }
```

We do a translation par `-\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```
5044 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5045 \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
5046 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
```

The dotted line will be extended if the user uses `margin` (or `left-margin` and `right-margin`).

The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn’t).

```
\begin{bNiceMatrix}
```

```
1 & 2 & 3 & 4 \\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
```

```
1 & 2 & 3 & 4 \\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

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

```
5047 \@@_qpoint:n { col - 1 }
5048 \dim_set:Nn \l_@@_x_initial_dim
5049 {
5050   \pgf@x +
```

We do a reduction by `\arraycolsep` for the environments with delimiters (and not for the other).

```
5051 \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
5052 - \l_@@_left_margin_dim
5053 }
5054 \@@_qpoint:n { col - \@@_succ:n \c@jCol }
5055 \dim_set:Nn \l_@@_x_final_dim
5056 {
5057   \pgf@x -
```

```

5058     \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
5059     + \l_@@_right_margin_dim
5060 }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by $0.5 \text{ } \l_@@_inter_dots_dim$ is *ad hoc* for a better result.

```

5061     \tl_if_eq:NnF \g_@@_left_delim_tl (
5062     { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
5063     \tl_if_eq:NnF \g_@@_right_delim_tl )
5064     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }

```

Up to now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

5065     \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
5066     \@@_draw_line:
5067 }

```

Vertical dotted lines

```

5068 \cs_new_protected:Npn \@@_vdottedline:n #1
5069 {
5070     \bool_set_true:N \l_@@_initial_open_bool
5071     \bool_set_true:N \l_@@_final_open_bool

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”.

```

5072     \bool_if:NTF \c_@@_tikz_loaded_bool
5073     {
5074         \tikzpicture
5075         \@@_vdottedline_i:n { #1 }
5076         \endtikzpicture
5077     }
5078     {
5079         \pgfpicture
5080         \@@_vdottedline_i:n { #1 }
5081         \endpgfpicture
5082     }
5083 }

```

```

5084 \cs_new_protected:Npn \@@_vdottedline_i:n #1
5085 {

```

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.

```

5086     \CT@arc@
5087     \pgfrememberpicturepositiononpagetrue
5088     \@@_qpoint:n { col - \int_eval:n { #1 + 1 } }

```

We do a translation par $-\l_@@_radius_dim$ because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```

5089     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
5090     \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
5091     \@@_qpoint:n { row - 1 }

```

We arbitrary decrease the height of the dotted line by a quantity equal to $\l_@@_inter_dots_dim$ in order to improve the visual impact.

```

5092     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
5093     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
5094     \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }

```

Up to now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

5095     \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
5096     \@@_draw_line:
5097 }

```

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.

```
5098 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
5099 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5100 {
5101   auto-columns-width .code:n =
5102   {
5103     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5104     \dim_gzero_new:N \g_@@_max_cell_width_dim
5105     \bool_set_true:N \l_@@_auto_columns_width_bool
5106   }
5107 }

5108 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
5109 {
5110   \int_gincr:N \g_@@_NiceMatrixBlock_int
5111   \dim_zero:N \l_@@_columns_width_dim
5112   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5113   \bool_if:NT \l_@@_block_auto_columns_width_bool
5114   {
5115     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5116     {
5117       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
5118       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5119     }
5120   }
5121 }
```

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).

```
5122 {
5123   \bool_if:NT \l_@@_block_auto_columns_width_bool
5124   {
5125     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5126     \iow_shipout:Nx \@mainaux
5127     {
5128       \cs_gset:cpn
5129       { @@_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.

```
5130       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5131     }
5132     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5133   }
5134 }
```

The extra nodes

First, two variants of the functions \dim_min:nn and \dim_max:nn.

```
5135 \cs_generate_variant:Nn \dim_min:nn { v n }
5136 \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).

```

5137 \cs_new_protected:Npn \@@_create_extra_nodes:
5138 {
5139   \bool_if:NTF \l_@@_medium_nodes_bool
5140   {
5141     \bool_if:NTF \l_@@_large_nodes_bool
5142     \@@_create_medium_and_large_nodes:
5143     \@@_create_medium_nodes:
5144   }
5145   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5146 }

```

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`.

```

5147 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5148 {
5149   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5150   {
5151     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
5152     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
5153     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
5154     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
5155   }
5156   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5157   {
5158     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
5159     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
5160     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
5161     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
5162   }

```

We begin the two nested loops over the rows and the columns of the array.

```

5163   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5164   {
5165     \int_step_variable:nnNn
5166     \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.

```

5167     {
5168       \cs_if_exist:cT
5169       { 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`.

```

5170     {
5171       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5172       \dim_set:cn { l_@@_row\_@@_i: _min_dim}

```

```

5173         { \dim_min:vn { l_@@_row _ @@_i: _min_dim } \pgf@y }
5174     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { @@_i: - @@_j: }
5175     {
5176         \dim_set:cn { l_@@_column _ @@_j: _min_dim }
5177         { \dim_min:vn { l_@@_column _ @@_j: _min_dim } \pgf@x }
5178     }

```

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`.

```

5179     \pgfpointanchor { @@_env: - @@_i: - @@_j: } { north-east }
5180     \dim_set:cn { l_@@_row _ @@_i: _ max_dim }
5181     { \dim_max:vn { l_@@_row _ @@_i: _ max_dim } \pgf@y }
5182     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { @@_i: - @@_j: }
5183     {
5184         \dim_set:cn { l_@@_column _ @@_j: _ max_dim }
5185         { \dim_max:vn { l_@@_column _ @@_j: _ max_dim } \pgf@x }
5186     }
5187 }
5188 }
5189 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

5190     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int @@_i:
5191     {
5192         \dim_compare:nNnT
5193         { \dim_use:c { l_@@_row _ @@_i: _ min _ dim } } = \c_max_dim
5194         {
5195             @@_qpoint:n { row - @@_i: - base }
5196             \dim_set:cn { l_@@_row _ @@_i: _ max _ dim } \pgf@y
5197             \dim_set:cn { l_@@_row _ @@_i: _ min _ dim } \pgf@y
5198         }
5199     }
5200     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int @@_j:
5201     {
5202         \dim_compare:nNnT
5203         { \dim_use:c { l_@@_column _ @@_j: _ min _ dim } } = \c_max_dim
5204         {
5205             @@_qpoint:n { col - @@_j: }
5206             \dim_set:cn { l_@@_column _ @@_j: _ max _ dim } \pgf@x
5207             \dim_set:cn { l_@@_column _ @@_j: _ min _ dim } \pgf@x
5208         }
5209     }
5210 }

```

Here is the command `@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

5211 \cs_new_protected:Npn @@_create_medium_nodes:
5212 {
5213     \pgfpicture
5214     \pgfrememberpicturepositiononpagetrue
5215     \pgf@relevantforpicturesizefalse
5216     @@_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”.

```

5217     \tl_set:Nn \l_@@_suffix_tl { -medium }
5218     @@_create_nodes:
5219     \endpgfpicture
5220 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁶⁴. 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:`.

```

5221 \cs_new_protected:Npn \@@_create_large_nodes:
5222 {
5223   \pgfpicture
5224     \pgfrememberpicturepositiononpagetrue
5225     \pgf@relevantforpicturesizefalse
5226     \@@_computations_for_medium_nodes:
5227     \@@_computations_for_large_nodes:
5228     \tl_set:Nn \l_@@_suffix_tl { - large }
5229     \@@_create_nodes:
5230   \endpgfpicture
5231 }

5232 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5233 {
5234   \pgfpicture
5235     \pgfrememberpicturepositiononpagetrue
5236     \pgf@relevantforpicturesizefalse
5237     \@@_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”.

```

5238   \tl_set:Nn \l_@@_suffix_tl { - medium }
5239   \@@_create_nodes:
5240   \@@_computations_for_large_nodes:
5241   \tl_set:Nn \l_@@_suffix_tl { - large }
5242   \@@_create_nodes:
5243   \endpgfpicture
5244 }

```

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.

```

5245 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5246 {
5247   \int_set:Nn \l_@@_first_row_int 1
5248   \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`.

```

5249   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5250   {
5251     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5252     {
5253       (
5254         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5255         \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5256       )
5257       / 2
5258     }
5259     \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5260     { l_@@_row _ \@@_i: _ min _ dim }
5261   }
5262   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5263   {
5264     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5265     {

```

⁶⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

5266      (
5267      \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5268      \dim_use:c
5269      { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5270      )
5271      / 2
5272      }
5273      \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5274      { l_@@_column _ \@@_j: _ max _ dim }
5275      }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

5276      \dim_sub:cn
5277      { l_@@_column _ 1 _ min _ dim }
5278      \l_@@_left_margin_dim
5279      \dim_add:cn
5280      { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5281      \l_@@_right_margin_dim
5282      }

```

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).

```

5283 \cs_new_protected:Npn \@@_create_nodes:
5284 {
5285   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5286   {
5287     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5288     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

5289       \@@_pgf_rect_node:nnnnn
5290       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5291       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
5292       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
5293       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
5294       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
5295       \str_if_empty:NF \l_@@_name_str
5296       {
5297         \pgfnodealias
5298         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5299         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5300       }
5301     }
5302   }

```

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 .

```

5303   \seq_mapthread_function:NNN
5304   \g_@@_multicolumn_cells_seq
5305   \g_@@_multicolumn_sizes_seq
5306   \@@_node_for_multicolumn:nn
5307   }

```

```

5308 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5309 {
5310   \cs_set_nopar:Npn \@@_i: { #1 }
5311   \cs_set_nopar:Npn \@@_j: { #2 }
5312 }

```

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”).

```

5313 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5314 {
5315   \@@_extract_coords_values: #1 \q_stop
5316   \@@_pgf_rect_node:nnnnn
5317     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5318     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
5319     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
5320     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
5321     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
5322   \str_if_empty:NF \l_@@_name_str
5323   {
5324     \pgfnodealias
5325       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5326       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
5327   }
5328 }

```

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.

```

5329 \keys_define:nn { NiceMatrix / Block / FirstPass }
5330 {
5331   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5332   l .value_forbidden:n = true ,
5333   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5334   r .value_forbidden:n = true ,
5335   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5336   c .value_forbidden:n = true ,
5337   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5338   L .value_forbidden:n = true ,
5339   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5340   R .value_forbidden:n = true ,
5341   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5342   C .value_forbidden:n = true ,
5343   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5344   t .value_forbidden:n = true ,
5345   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5346   b .value_forbidden:n = true ,
5347   color .tl_set:N = \l_@@_color_tl ,
5348   color .value_required:n = true ,
5349 }

```

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.

```

5350 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
5351 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not be provided by the user, you use 1-1 (that is to say a block of only one cell).

```

5352   \peek_remove_spaces:n
5353   {
5354     \tl_if_blank:nTF { #2 }

```



```

5355         { \@@_Block_i 1-1 \q_stop }
5356         { \@@_Block_i #2 \q_stop }
5357     { #1 } { #3 } { #4 }
5358 }
5359 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5360 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

Now, the arguments have been extracted: $\#1$ is i (the number of rows of the block), $\#2$ is j (the number of columns of the block), $\#3$ is the list of key-values, $\#4$ are the tokens to put before the math mode and the beginning of the small array of the block and $\#5$ is the label of the block.

```

5361 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5362 {

```

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).

```

5363     \bool_lazy_or:nnTF
5364     { \tl_if_blank_p:n { #1 } }
5365     { \str_if_eq_p:nn { #1 } { * } }
5366     { \int_set:Nn \l_tmpa_int { 100 } }
5367     { \int_set:Nn \l_tmpa_int { #1 } }
5368     \bool_lazy_or:nnTF
5369     { \tl_if_blank_p:n { #2 } }
5370     { \str_if_eq_p:nn { #2 } { * } }
5371     { \int_set:Nn \l_tmpb_int { 100 } }
5372     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5373     \int_compare:nNnTF \l_tmpb_int = 1
5374     {
5375         \str_if_empty:NTF \l_@@_hpos_cell_str
5376         { \str_set:Nn \l_@@_hpos_block_str c }
5377         { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
5378     }
5379     { \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.

```

5380     \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
5381     \tl_set:Nx \l_tmpa_tl
5382     {
5383         { \int_use:N \c@iRow }
5384         { \int_use:N \c@jCol }
5385         { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5386         { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5387     }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

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

If the block is mono-column or mono-row, we have a special treatment. That's why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by currying).

```

5388     \bool_if:nTF
5389     {
5390     (

```

```

5391 \int_compare_p:nNn { \l_tmpa_int } = 1
5392 ||
5393 \int_compare_p:nNn { \l_tmpb_int } = 1
5394 )
5395 && ! \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.

```

5396 && ! \l_@@_X_column_bool
5397 }
5398 { \exp_args:Nxx \@@_Block_iv:nnnnn }
5399 { \exp_args:Nxx \@@_Block_v:nnnnn }
5400 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5401 }

```

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).

```

5402 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5403 {
5404   \int_gincr:N \g_@@_block_box_int
5405   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5406   {
5407     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5408     {
5409       \@@_actually_diagbox:nnnnnn
5410       { \int_use:N \c@iRow }
5411       { \int_use:N \c@jCol }
5412       { \int_eval:n { \c@iRow + #1 - 1 } }
5413       { \int_eval:n { \c@jCol + #2 - 1 } }
5414       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5415     }
5416   }
5417   \box_gclear_new:c
5418   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
5419   \hbox_gset:cn
5420   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
5421   {

```

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}`).

```

5422 \tl_if_empty:NTF \l_@@_color_tl
5423 { \int_compare:nNnT { #2 } = 1 \set@color }
5424 { \color { \l_@@_color_tl } }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even if it has yet been used in the beginning of the cell where the command `\Block` has been issued because we want to be able to take into account a potential instruction of color of the font in `\g_@@_row_style_tl`.

```

5425 \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5426 \group_begin:
5427 \cs_set:Npn \arraystretch { 1 }
5428 \dim_set_eq:NN \extrarowheight \c_zero_dim
5429 #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).

```

5430 \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
5431 \bool_if:NTF \l_@@_NiceTabular_bool
5432 {
5433   \bool_lazy_and:nnTF
5434   { \int_compare_p:nNn { #2 } = 1 }
5435   { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }

```

When the block is mono-column in a column with a fixed width (eg p{3cm}).

```

5436 {
5437   \begin { minipage } [ \l_@@_vpos_of_block_tl ]
5438   { \l_@@_col_width_dim }
5439   \str_case:Vn \l_@@_hpos_block_str
5440   {
5441     c \centering
5442     r \raggedleft
5443     l \raggedright
5444   }
5445   #5
5446   \end { minipage }
5447 }
5448 {
5449   \use:x
5450   {
5451     \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5452     { @ { } \l_@@_hpos_block_str @ { } }
5453   }
5454   #5
5455   \end { tabular }
5456 }
5457 }
5458 {
5459   \c_math_toggle_token
5460   \use:x
5461   {
5462     \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5463     { @ { } \l_@@_hpos_block_str @ { } }
5464   }
5465   #5
5466   \end { array }
5467   \c_math_toggle_token
5468 }
5469 \group_end:
5470 }
5471 \bool_if:NT \g_@@_rotate_bool
5472 {
5473   \box_grotate:cn
5474   { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5475   { 90 }
5476   \bool_gset_false:N \g_@@_rotate_bool
5477 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5478 \int_compare:nNnT { #2 } = 1
5479 {
5480   \dim_gset:Nn \g_@@_blocks_wd_dim
5481   {
5482     \dim_max:nn
5483     \g_@@_blocks_wd_dim
5484     {
5485       \box_wd:c

```

```

5486         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5487     }
5488 }
5489 }

```

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.

```

5490 \int_compare:nNnT { #1 } = 1
5491 {
5492     \dim_gset:Nn \g_@@_blocks_ht_dim
5493     {
5494         \dim_max:nn
5495         \g_@@_blocks_ht_dim
5496         {
5497             \box_ht:c
5498             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5499         }
5500     }
5501     \dim_gset:Nn \g_@@_blocks_dp_dim
5502     {
5503         \dim_max:nn
5504         \g_@@_blocks_dp_dim
5505         {
5506             \box_dp:c
5507             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5508         }
5509     }
5510 }
5511 \seq_gput_right:Nx \g_@@_blocks_seq
5512 {
5513     \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.

```

5514     { \exp_not:n { #3 } , \l_@@_hpos_block_str }
5515     {
5516         \box_use_drop:c
5517         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5518     }
5519 }
5520 }

```

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.

```

5521 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5522 {
5523     \seq_gput_right:Nx \g_@@_blocks_seq
5524     {
5525         \l_tmpa_tl
5526         { \exp_not:n { #3 } }
5527         \exp_not:n
5528         {
5529             {
5530                 \bool_if:NTF \l_@@_NiceTabular_bool
5531                 {
5532                     \group_begin:
5533                     \cs_set:Npn \arraystretch { 1 }
5534                     \dim_set_eq:NN \extrarowheight \c_zero_dim
5535                     #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).

```

5536         \bool_if:NT \g_@@_rotate_bool
5537         { \str_set:Nn \l_@@_hpos_block_str c }
5538     \use:x
5539     {
5540         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5541         { @ { } \l_@@_hpos_block_str @ { } }
5542     }
5543     #5
5544     \end { tabular }
5545     \group_end:
5546 }
5547 {
5548     \group_begin:
5549     \cs_set:Npn \arraystretch { 1 }
5550     \dim_set_eq:NN \extrarowheight \c_zero_dim
5551     #4
5552     \bool_if:NT \g_@@_rotate_bool
5553     { \str_set:Nn \l_@@_hpos_block_str c }
5554     \c_math_toggle_token
5555     \use:x
5556     {
5557         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5558         { @ { } \l_@@_hpos_block_str @ { } }
5559     }
5560     #5
5561     \end { array }
5562     \c_math_toggle_token
5563     \group_end:
5564 }
5565 }
5566 }
5567 }
5568 }
```

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).

```

5569 \keys_define:nn { NiceMatrix / Block / SecondPass }
5570 {
5571     tikz .code:n =
5572         \bool_if:NTF \c_@@_tikz_loaded_bool
5573         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
5574         { \@@_error:n { tikz-key-without~tikz } } ,
5575     tikz .value_required:n = true ,
5576     fill .tl_set:N = \l_@@_fill_tl ,
5577     fill .value_required:n = true ,
5578     draw .tl_set:N = \l_@@_draw_tl ,
5579     draw .default:n = default ,
5580     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5581     rounded-corners .default:n = 4 pt ,
5582     color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5583     color .value_required:n = true ,
5584     borders .clist_set:N = \l_@@_borders_clist ,
5585     borders .value_required:n = true ,
5586     hvlines .bool_set:N = \l_@@_hvlines_block_bool ,
5587     hvlines .default:n = true ,
5588     line-width .dim_set:N = \l_@@_line_width_dim ,
```

```

5589   line-width .value_required:n = true ,
5590   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5591   l .value_forbidden:n = true ,
5592   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5593   r .value_forbidden:n = true ,
5594   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5595   c .value_forbidden:n = true ,
5596   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
5597             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5598   L .value_forbidden:n = true ,
5599   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
5600             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5601   R .value_forbidden:n = true ,
5602   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
5603             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5604   C .value_forbidden:n = true ,
5605   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5606   t .value_forbidden:n = true ,
5607   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5608   b .value_forbidden:n = true ,
5609   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
5610 }

```

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.

```

5611 \cs_new_protected:Npn \@@_draw_blocks:
5612 {
5613   \cs_set_eq:NN \ialign \@@_old_ialign:
5614   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
5615 }
5616 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5617 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5618   \int_zero_new:N \l_@@_last_row_int
5619   \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.

```

5620   \int_compare:nNnTF { #3 } > { 99 }
5621   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
5622   { \int_set:Nn \l_@@_last_row_int { #3 } }
5623   \int_compare:nNnTF { #4 } > { 99 }
5624   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
5625   { \int_set:Nn \l_@@_last_col_int { #4 } }
5626   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
5627   {
5628     \int_compare:nTF
5629     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
5630     {
5631       \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
5632       \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
5633       \group_begin:
5634       \globaldefs = 1
5635       \@@_msg_redirect_name:nn { columns~not~used } { none }
5636       \group_end:

```

```

5637     }
5638     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5639   }
5640   {
5641     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
5642     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5643     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
5644   }
5645 }
5646 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
5647 {

```

The sequence of the positions of the blocks will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

5648   \seq_gput_left:Nn \g_@@_pos_of_blocks_seq { { #1 } { #2 } { #3 } { #4 } }

```

The group is for the keys.

```

5649   \group_begin:
5650   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
5651   \bool_if:NT \l_@@_hvlines_block_bool
5652   {
5653     \tl_gput_right:Nx \g_nicematrix_code_after_tl
5654     {
5655       \@@_hvlines_block:nnn
5656       { \exp_not:n { #5 } }
5657       { #1 - #2 }
5658       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5659     }
5660   }
5661   \tl_if_empty:NF \l_@@_draw_tl
5662   {
5663     \tl_gput_right:Nx \g_nicematrix_code_after_tl
5664     {
5665       \@@_stroke_block:nnn
5666       { \exp_not:n { #5 } }
5667       { #1 - #2 }
5668       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5669     }
5670     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
5671     { { #1 } { #2 } { #3 } { #4 } }
5672   }
5673   \clist_if_empty:NF \l_@@_borders_clist
5674   {
5675     \tl_gput_right:Nx \g_nicematrix_code_after_tl
5676     {
5677       \@@_stroke_borders_block:nnn
5678       { \exp_not:n { #5 } }
5679       { #1 - #2 }
5680       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5681     }
5682   }
5683   \tl_if_empty:NF \l_@@_fill_tl
5684   {

```

The command `\@@_extract_brackets` will extract the potential specification of color space at the beginning of `\l_@@_fill_tl` and store it in `\l_tmpa_tl` and store the color itself in `\l_tmpb_tl`.

```

5685     \exp_last_unbraced:NV \@@_extract_brackets \l_@@_fill_tl \q_stop
5686     \tl_gput_right:Nx \g_nicematrix_code_before_tl
5687     {
5688       \exp_not:N \roundedrectanglecolor
5689       [ \l_tmpa_tl ]
5690       { \exp_not:V \l_tmpb_tl }

```

```

5691         { #1 - #2 }
5692         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5693         { \dim_use:N \l_@@_rounded_corners_dim }
5694     }
5695 }
5696 \seq_if_empty:NF \l_@@_tikz_seq
5697 {
5698     \tl_gput_right:Nx \g_nicematrix_code_before_tl
5699     {
5700         \@@_block_tikz:nnnnn
5701         { #1 }
5702         { #2 }
5703         { \int_use:N \l_@@_last_row_int }
5704         { \int_use:N \l_@@_last_col_int }
5705         { \seq_use:Nn \l_@@_tikz_seq { , } }
5706     }
5707 }
5708 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5709 {
5710     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5711     {
5712         \@@_actually_diagbox:nnnnnn
5713         { #1 }
5714         { #2 }
5715         { \int_use:N \l_@@_last_row_int }
5716         { \int_use:N \l_@@_last_col_int }
5717         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5718     }
5719 }
5720 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
5721 \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.

```

5722 \pgfpicture
5723 \pgfrememberpicturepositiononpagetrue
5724 \pgf@relevantforpicturesizefalse
5725 \@@_qpoint:n { row - #1 }
5726 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5727 \@@_qpoint:n { col - #2 }
5728 \dim_set_eq:NN \l_tmpb_dim \pgf@x

```



```

5729 \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
5730 \dim_set_eq:NN \l_tmpc_dim \pgf@y
5731 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5732 \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.

```

5733 \begin { pgfscope }
5734 \@@_pgf_rect_node:nnnnn
5735 { \@@_env: - #1 - #2 - block }
5736 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
5737 \end { pgfscope }

```

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.

```

5738 \bool_if:NF \l_@@_hpos_of_block_cap_bool
5739 {
5740 \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.

```

5741 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5742 {

```

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.

```

5743 \cs_if_exist:cT
5744 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5745 {
5746 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5747 {
5748 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
5749 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
5750 }
5751 }
5752 }

```

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.

```

5753 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
5754 {
5755 \@@_qpoint:n { col - #2 }
5756 \dim_set_eq:NN \l_tmpb_dim \pgf@x
5757 }
5758 \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
5759 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5760 {
5761 \cs_if_exist:cT
5762 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5763 {
5764 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5765 {
5766 \pgfpointanchor
5767 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5768 { east }
5769 \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
5770 }
5771 }
5772 }
5773 \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
5774 {

```

```

5775         \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5776         \dim_set_eq:NN \l_tmpd_dim \pgf@x
5777     }
5778     \@@_pgf_rect_node:nnnnn
5779     { \@@_env: - #1 - #2 - block - short }
5780     \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpe_dim
5781 }

```

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.

```

5782     \bool_if:NT \l_@@_medium_nodes_bool
5783     {
5784         \@@_pgf_rect_node:nnn
5785         { \@@_env: - #1 - #2 - block - medium }
5786         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
5787         {
5788             \pgfpointanchor
5789             { \@@_env:
5790               - \int_use:N \l_@@_last_row_int
5791               - \int_use:N \l_@@_last_col_int - medium
5792             }
5793             { south-east }
5794         }
5795     }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

5796     \int_compare:nNnTF { #1 } = { #3 }
5797     {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

5798         \int_compare:nNnTF { #1 } = 0
5799         { \l_@@_code_for_first_row_tl }
5800         {
5801             \int_compare:nNnT { #1 } = \l_@@_last_row_int
5802             \l_@@_code_for_last_row_tl
5803         }

```

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`.

```

5804         \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

5805         \pgfpointanchor
5806         {
5807             \@@_env: - #1 - #2 - block
5808             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
5809         }
5810         {
5811             \str_case:Vn \l_@@_hpos_block_str
5812             {
5813                 c { center }
5814                 l { west }
5815                 r { east }
5816             }
5817         }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

5818         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
5819         \pgfset { inner~sep = \c_zero_dim }
5820         \pgfnode
5821         { rectangle }
5822         {
5823             \str_case:Vn \l_@@_hpos_block_str
5824             {

```

```

5825         c { base }
5826         l { base~west }
5827         r { base~east }
5828     }
5829 }
5830 { \box_use_drop:N \l_@@_cell_box } { } { }
5831 }

```

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).

```

5832 {

```

If we are in the first column, we must put the block as if it was with the key r.

```

5833     \int_compare:nNnT { #2 } = 0
5834     { \str_set:Nn \l_@@_hpos_block_str r }
5835     \bool_if:nT \g_@@_last_col_found_bool
5836     {
5837         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5838         { \str_set:Nn \l_@@_hpos_block_str l }
5839     }
5840     \pgftransformshift
5841     {
5842         \pgfpointanchor
5843         {
5844             \@@_env: - #1 - #2 - block
5845             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
5846         }
5847         {
5848             \str_case:Vn \l_@@_hpos_block_str
5849             {
5850                 c { center }
5851                 l { west }
5852                 r { east }
5853             }
5854         }
5855     }
5856     \pgfset { inner-sep = \c_zero_dim }
5857     \pgfnode
5858     { rectangle }
5859     {
5860         \str_case:Vn \l_@@_hpos_block_str
5861         {
5862             c { center }
5863             l { west }
5864             r { east }
5865         }
5866     }
5867     { \box_use_drop:N \l_@@_cell_box } { } { }
5868 }
5869 \endpgfpicture
5870 \group_end:
5871 }

```

```

5872 \NewDocumentCommand \@@_extract_brackets { 0 { } }
5873 {
5874     \tl_set:Nn \l_tmpa_tl { #1 }
5875     \@@_store_in_tmpb_tl
5876 }
5877 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
5878 { \tl_set:Nn \l_tmpb_tl { #1 } }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax $i-j$) and the third is the last cell of the block (with the same syntax).

```

5879 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
5880 {
5881   \group_begin:
5882   \tl_clear:N \l_@@_draw_tl
5883   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5884   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
5885   \pgfpicture
5886   \pgfrememberpicturepositiononpagetrue
5887   \pgf@relevantforpicturesizefalse
5888   \tl_if_empty:NF \l_@@_draw_tl
5889   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

5890     \str_if_eq:VnTF \l_@@_draw_tl { default }
5891     { \CT@arc@ }
5892     { \exp_args:NW \pgfsetstrokecolor \l_@@_draw_tl }
5893   }
5894   \pgfsetcornersarced
5895   {
5896     \pgfpoint
5897     { \dim_use:N \l_@@_rounded_corners_dim }
5898     { \dim_use:N \l_@@_rounded_corners_dim }
5899   }
5900   \@@_cut_on_hyphen:w #2 \q_stop
5901   \bool_lazy_and:nnT
5902   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
5903   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
5904   {
5905     \@@_qpoint:n { row - \l_tmpa_tl }
5906     \dim_set:Nn \l_tmpb_dim { \pgf@y }
5907     \@@_qpoint:n { col - \l_tmpb_tl }
5908     \dim_set:Nn \l_tmpc_dim { \pgf@x }
5909     \@@_cut_on_hyphen:w #3 \q_stop
5910     \int_compare:nNnT \l_tmpa_tl > \c@iRow
5911     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
5912     \int_compare:nNnT \l_tmpb_tl > \c@jCol
5913     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5914     \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
5915     \dim_set:Nn \l_tmpa_dim { \pgf@y }
5916     \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
5917     \dim_set:Nn \l_tmpd_dim { \pgf@x }
5918     \pgfpathrectanglecorners
5919     { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5920     { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5921     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

5922     \pgfusepath { stroke }
5923   }
5924   \endpgfpicture
5925   \group_end:
5926 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

5927 \keys_define:nn { NiceMatrix / BlockStroke }
5928 {
5929   color .tl_set:N = \l_@@_draw_tl ,
5930   draw .tl_set:N = \l_@@_draw_tl ,
5931   draw .default:n = default ,
5932   line-width .dim_set:N = \l_@@_line_width_dim ,

```

```

5933   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5934   rounded-corners .default:n = 4 pt
5935 }

```

The first argument of `\@@_hvlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax $i-j$) and the third is the last cell of the block (with the same syntax).

```

5936 \cs_new_protected:Npn \@@_hvlines_block:nnn #1 #2 #3
5937 {
5938   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5939   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5940   \@@_cut_on_hyphen:w #2 \q_stop
5941   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5942   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5943   \@@_cut_on_hyphen:w #3 \q_stop
5944   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5945   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5946   \pgfpicture
5947   \pgfrememberpicturepositiononpagetrue
5948   \pgf@relevantforpicturesizefalse
5949   \CT@arc@
5950   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

First, the vertical rules.

```

5951   \@@_qpoint:n { row - \l_tmpa_tl }
5952   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5953   \@@_qpoint:n { row - \l_tmpc_tl }
5954   \dim_set_eq:NN \l_tmpb_dim \pgf@y
5955   \int_step_inline:nnn \l_tmpd_tl \l_tmpb_tl
5956   {
5957     \@@_qpoint:n { col - ##1 }
5958     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpa_dim }
5959     \pgfpathlineto { \pgfpoint \pgf@x \l_tmpb_dim }
5960     \pgfusepathqstroke
5961   }

```

Now, the horizontal rules.

```

5962   \@@_qpoint:n { col - \l_tmpb_tl }
5963   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
5964   \@@_qpoint:n { col - \l_tmpd_tl }
5965   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \arrayrulewidth }
5966   \int_step_inline:nnn \l_tmpc_tl \l_tmpa_tl
5967   {
5968     \@@_qpoint:n { row - ##1 }
5969     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5970     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5971     \pgfusepathqstroke
5972   }
5973   \endpgfpicture
5974 }

```

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).

```

5975 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
5976 {
5977   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5978   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5979   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
5980   { \@@_error:n { borders~forbidden } }
5981   {
5982     \clist_map_inline:Nn \l_@@_borders_clist
5983     {

```

```

5984         \clist_if_in:nnF { top , bottom , left , right } { ##1 }
5985         { \@@_error:nn { bad-border } { ##1 } }
5986     }
5987     \@@_cut_on_hyphen:w #2 \q_stop
5988     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5989     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5990     \@@_cut_on_hyphen:w #3 \q_stop
5991     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5992     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5993     \pgfpicture
5994     \pgfrememberpicturepositiononpagetrue
5995     \pgf@relevantforpicturesizefalse
5996     \CT@arc@
5997     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
5998     \clist_if_in:NnT \l_@@_borders_clist { right }
5999     { \@@_stroke_vertical:n \l_tmpb_tl }
6000     \clist_if_in:NnT \l_@@_borders_clist { left }
6001     { \@@_stroke_vertical:n \l_tmpd_tl }
6002     \clist_if_in:NnT \l_@@_borders_clist { bottom }
6003     { \@@_stroke_horizontal:n \l_tmpa_tl }
6004     \clist_if_in:NnT \l_@@_borders_clist { top }
6005     { \@@_stroke_horizontal:n \l_tmpc_tl }
6006     \endpgfpicture
6007 }
6008 }

```

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).

```

6009 \cs_new_protected:Npn \@@_stroke_vertical:n #1
6010 {
6011     \@@_qpoint:n \l_tmpc_tl
6012     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6013     \@@_qpoint:n \l_tmpa_tl
6014     \dim_set:Nn \l_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6015     \@@_qpoint:n { #1 }
6016     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
6017     \pgfpathlineto { \pgfpoint \pgf@x \l_tmpc_dim }
6018     \pgfusepathqstroke
6019 }

```

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).

```

6020 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
6021 {
6022     \@@_qpoint:n \l_tmpd_tl
6023     \clist_if_in:NnTF \l_@@_borders_clist { left }
6024     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
6025     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
6026     \@@_qpoint:n \l_tmpb_tl
6027     \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
6028     \@@_qpoint:n { #1 }
6029     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
6030     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6031     \pgfusepathqstroke
6032 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

6033 \keys_define:nn { NiceMatrix / BlockBorders }
6034 {
6035     borders .clist_set:N = \l_@@_borders_clist ,
6036     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6037     rounded-corners .default:n = 4 pt ,
6038     line-width .dim_set:N = \l_@@_line_width_dim

```

```
6039 }
```

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.

```
6040 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
6041 {
6042   \begin { tikzpicture }
6043   \clist_map_inline:nn { #5 }
6044     {
6045       \path [ ##1 ]
6046         ( #1 -| #2 ) rectangle ( \@@_succ:n { #3 } -| \@@_succ:n { #4 } ) ;
6047     }
6048   \end { tikzpicture }
6049 }
```

How to draw the dotted lines transparently

```
6050 \cs_set_protected:Npn \@@_renew_matrix:
6051 {
6052   \RenewDocumentEnvironment { pmatrix } { } { }
6053   { \pNiceMatrix }
6054   { \endpNiceMatrix }
6055   \RenewDocumentEnvironment { vmatrix } { } { }
6056   { \vNiceMatrix }
6057   { \endvNiceMatrix }
6058   \RenewDocumentEnvironment { Vmatrix } { } { }
6059   { \VNiceMatrix }
6060   { \endVNiceMatrix }
6061   \RenewDocumentEnvironment { bmatrix } { } { }
6062   { \bNiceMatrix }
6063   { \endbNiceMatrix }
6064   \RenewDocumentEnvironment { Bmatrix } { } { }
6065   { \BNiceMatrix }
6066   { \endBNiceMatrix }
6067 }
```

Automatic arrays

```
6068 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
6069 {
6070   \int_set:Nn \l_@@_nb_rows_int { #1 }
6071   \int_set:Nn \l_@@_nb_cols_int { #2 }
6072 }
```

We will extract the potential keys `l`, `r` and `c` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```
6073 \keys_define:nn { NiceMatrix / Auto }
6074 {
6075   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
6076   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
6077   c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
6078 }
6079 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
6080 {
6081   \int_zero_new:N \l_@@_nb_rows_int
6082   \int_zero_new:N \l_@@_nb_cols_int
6083   \@@_set_size:n #4 \q_stop

```

The group is for the protection of `\l_@@_type_of_col_tl`.

```
6084 \group_begin:
```

```

6085 \tl_set:Nn \l_@@_type_of_col_tl c
6086 \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
6087 \use:x
6088 {
6089   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
6090   { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
6091   [ \exp_not:V \l_tmpa_tl ]
6092 }
6093 \int_compare:nNnT \l_@@_first_row_int = 0
6094 {
6095   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6096   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6097   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6098 }
6099 \prg_replicate:nn \l_@@_nb_rows_int
6100 {
6101   \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).

```

6102   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
6103   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6104 }
6105 \int_compare:nNnT \l_@@_last_row_int > { -2 }
6106 {
6107   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6108   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6109   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6110 }
6111 \end { NiceArrayWithDelims }
6112 \group_end:
6113 }
6114 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6115 {
6116   \cs_set_protected:cpn { #1 AutoNiceMatrix }
6117   {
6118     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6119     \AutoNiceMatrixWithDelims { #2 } { #3 }
6120   }
6121 }
6122 \@@_define_com:nnn p ( )
6123 \@@_define_com:nnn b [ ]
6124 \@@_define_com:nnn v | |
6125 \@@_define_com:nnn V \l \l
6126 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

6127 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
6128 {
6129   \group_begin:
6130   \bool_set_true:N \l_@@_NiceArray_bool
6131   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6132   \group_end:
6133 }

```

The redefinition of the command \dotfill

```

6134 \cs_set_eq:NN \@@_old_dotfill \dotfill
6135 \cs_new_protected:Npn \@@_dotfill:
6136 {

```


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}`).

```

6137 \@@_old_dotfill
6138 \bool_if:NT \l_@@_NiceTabular_bool
6139 { \group_insert_after:N \@@_dotfill_ii: }
6140 { \group_insert_after:N \@@_dotfill_i: }
6141 }
6142 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
6143 \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`.

```

6144 \cs_new_protected:Npn \@@_dotfill_iii:
6145 { \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.

```

6146 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6147 {
6148   \tl_gput_right:Nx \g_@@_internal_code_after_tl
6149   {
6150     \@@_actually_diagbox:nnnnnn
6151     { \int_use:N \c_iRow }
6152     { \int_use:N \c_jCol }
6153     { \int_use:N \c_iRow }
6154     { \int_use:N \c_jCol }
6155     { \exp_not:n { #1 } }
6156     { \exp_not:n { #2 } }
6157   }

```

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`.

```

6158 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6159 {
6160   { \int_use:N \c_iRow }
6161   { \int_use:N \c_jCol }
6162   { \int_use:N \c_iRow }
6163   { \int_use:N \c_jCol }
6164 }
6165 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it’s possible to use `\diagbox` in a `\Block`). The two other are the elements to draw below and above the diagonal line.

```

6166 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
6167 {
6168   \pgfpicture
6169   \pgf@relevantforpicturesizefalse
6170   \pgfrememberpicturepositiononpagetrue
6171   \@@_qpoint:n { row - #1 }
6172   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6173   \@@_qpoint:n { col - #2 }
6174   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6175   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6176   \@@_qpoint:n { row - \@@_succ:n { #3 } }
6177   \dim_set_eq:NN \l_tmpc_dim \pgf@y
6178   \@@_qpoint:n { col - \@@_succ:n { #4 } }
6179   \dim_set_eq:NN \l_tmpd_dim \pgf@x

```

```

6180 \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
6181 {

```

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.

```

6182 \CT@arc@
6183 \pgfsetroundcap
6184 \pgfusepathqstroke
6185 }
6186 \pgfset { inner-sep = 1 pt }
6187 \pgfscope
6188 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
6189 \pgfnode { rectangle } { south-west }
6190 { \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }
6191 \endpgfscope
6192 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
6193 \pgfnode { rectangle } { north-east }
6194 { \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
6195 \endpgfpicture
6196 }

```

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.

```

6197 \keys_define:nn { NiceMatrix }
6198 {
6199   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6200   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6201 }
6202 \keys_define:nn { NiceMatrix / CodeAfter }
6203 {
6204   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6205   sub-matrix .value_required:n = true ,
6206   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6207   delimiters / color .value_required:n = true ,
6208   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6209   rules .value_required:n = true ,
6210   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
6211 }

```

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. 114.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

6212 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_i:n }

```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_i:n` which do *not* begin with `\omit` (and thus, the user will be able to use `\CodeAfter` without error and without the need to prefix by `\omit`).

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

6213 \cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
6214 {
6215   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6216   \@@_CodeAfter_ii:n
6217 }

```

We catch the argument of the command `\end` (in `#1`).

```
6218 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1
6219 {
```

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.

```
6220 \str_if_eq:eeTF \currentenvir { #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`.

```
6221 {
6222   \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6223   \@@_CodeAfter_i:n
6224 }
6225 }
```

The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_internal_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of columnn. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
6226 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
6227 {
6228   \pgfpicture
6229   \pgfrememberpicturepositiononpagetrue
6230   \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.

```
6231 \@@_qpoint:n { row - 1 }
6232 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6233 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
6234 \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).

```
6235 \bool_if:nTF { #3 }
6236 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6237 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6238 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6239 {
6240   \cs_if_exist:cT
6241   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6242   {
6243     \pgfpointanchor
6244     { \@@_env: - ##1 - #2 }
6245     { \bool_if:nTF { #3 } { west } { east } }
6246     \dim_set:Nn \l_tmpa_dim
6247     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6248   }
6249 }
```

Now we can put the delimiter with a node of PGF.

```

6250 \pgfset { inner~sep = \c_zero_dim }
6251 \dim_zero:N \nulldelimiterspace
6252 \pgftransformshift
6253 {
6254   \pgfpoint
6255   { \l_tmpa_dim }
6256   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6257 }
6258 \pgfnode
6259 { rectangle }
6260 { \bool_if:nTF { #3 } { east } { west } }
6261 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

6262   \nullfont
6263   \c_math_toggle_token
6264   \tl_if_empty:NF \l_@@_delimiters_color_tl
6265   { \color { \l_@@_delimiters_color_tl } }
6266   \bool_if:nTF { #3 } { \left #1 } { \left . }
6267   \vcenter
6268   {
6269     \nullfont
6270     \hrule \@height
6271     \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6272     \@depth \c_zero_dim
6273     \@width \c_zero_dim
6274   }
6275   \bool_if:nTF { #3 } { \right . } { \right #1 }
6276   \c_math_toggle_token
6277 }
6278 { }
6279 { }
6280 \endpgfpicture
6281 }

```

The command `\SubMatrix`

```

6282 \keys_define:nn { NiceMatrix / sub-matrix }
6283 {
6284   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6285   extra-height .value_required:n = true ,
6286   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6287   left-xshift .value_required:n = true ,
6288   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6289   right-xshift .value_required:n = true ,
6290   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6291   xshift .value_required:n = true ,
6292   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6293   delimiters / color .value_required:n = true ,
6294   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6295   slim .default:n = true ,
6296   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6297   hlines .default:n = all ,
6298   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6299   vlines .default:n = all ,
6300   hvlines .meta:n = { hlines, vlines } ,
6301   hvlines .value_forbidden:n = true ,
6302 }
6303 \keys_define:nn { NiceMatrix }
6304 {
6305   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6306   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,

```

```

6307 NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6308 NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6309 pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6310 NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6311 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

6312 \keys_define:nn { NiceMatrix / SubMatrix }
6313 {
6314   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6315   hlines .default:n = all ,
6316   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6317   vlines .default:n = all ,
6318   hvlines .meta:n = { hlines, vlines } ,
6319   hvlines .value_forbidden:n = true ,
6320   name .code:n =
6321     \tl_if_empty:nTF { #1 }
6322     { \@@_error:n { Invalid-name-format } }
6323     {
6324       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6325       {
6326         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6327         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
6328         {
6329           \str_set:Nn \l_@@_submatrix_name_str { #1 }
6330           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6331         }
6332       }
6333       { \@@_error:n { Invalid-name-format } }
6334     } ,
6335   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6336   rules .value_required:n = true ,
6337   code .tl_set:N = \l_@@_code_tl ,
6338   code .value_required:n = true ,
6339   name .value_required:n = true ,
6340   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
6341 }

6342 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
6343 {
6344   \peek_remove_spaces:n
6345   {
6346     \@@_cut_on_hyphen:w #3 \q_stop
6347     \tl_clear_new:N \l_tmpc_tl
6348     \tl_clear_new:N \l_tmpd_tl
6349     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6350     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6351     \@@_cut_on_hyphen:w #2 \q_stop
6352     \seq_gput_right:Nx \g_@@_submatrix_seq
6353     { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
6354     \tl_gput_right:Nn \g_@@_internal_code_after_tl
6355     { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6356   }
6357 }

```

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`.

```

6358 \AtBeginDocument
6359 {
6360   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
6361   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
6362   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
6363   {
6364     \peek_remove_spaces:n
6365     { \@@_sub_matrix:nnnnnnn
6366       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
6367   }
6368 }

6369 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6370 {
6371   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

6372 \tl_clear_new:N \l_@@_first_i_tl
6373 \tl_clear_new:N \l_@@_first_j_tl
6374 \tl_clear_new:N \l_@@_last_i_tl
6375 \tl_clear_new:N \l_@@_last_j_tl

```

The command `\@@_cut_on_hyphen:w` cuts on the hyphen an argument of the form $i-j$. The value of i is stored in `\l_tmpa_tl` and the value of j is stored in `\l_tmpb_tl`.

```

6376 \@@_cut_on_hyphen:w #2 \q_stop
6377 \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
6378 \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
6379 \@@_cut_on_hyphen:w #3 \q_stop
6380 \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
6381 \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
6382 \bool_lazy_or:nnTF
6383 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6384 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6385 { \@@_error:n { SubMatrix~too~large } }
6386 {
6387   \str_clear_new:N \l_@@_submatrix_name_str
6388   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6389   \pgfpicture
6390   \pgfrememberpicturepositiononpagetrue
6391   \pgf@relevantforpicturesizefalse
6392   \pgfset { inner~sep = \c_zero_dim }
6393   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6394   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

6395 \bool_if:NTF \l_@@_submatrix_slim_bool
6396 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6397 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6398 {
6399   \cs_if_exist:cT
6400   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6401   {
6402     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6403     \dim_set:Nn \l_@@_x_initial_dim
6404     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6405   }

```

```

6406         \cs_if_exist:cT
6407         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6408         {
6409             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6410             \dim_set:Nn \l_@@_x_final_dim
6411             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6412         }
6413     }
6414     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
6415     { \@@_error:nn { impossible-delimiter } { left } }
6416     {
6417         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
6418         { \@@_error:nn { impossible~delimiter } { right } }
6419         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
6420     }
6421     \endpgfpicture
6422 }
6423 \group_end:
6424 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

6425 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
6426 {
6427     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
6428     \dim_set:Nn \l_@@_y_initial_dim
6429     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
6430     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
6431     \dim_set:Nn \l_@@_y_final_dim
6432     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
6433     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
6434     {
6435         \cs_if_exist:cT
6436         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
6437         {
6438             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
6439             \dim_set:Nn \l_@@_y_initial_dim
6440             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6441         }
6442         \cs_if_exist:cT
6443         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6444         {
6445             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6446             \dim_set:Nn \l_@@_y_final_dim
6447             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6448         }
6449     }
6450     \dim_set:Nn \l_tmpa_dim
6451     {
6452         \l_@@_y_initial_dim - \l_@@_y_final_dim +
6453         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6454     }
6455     \dim_set_eq:NN \nulldelimiterspace \c_zero_dim

```

We will draw the rules in the `\SubMatrix`.

```

6456 \group_begin:
6457 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6458 \tl_if_empty:NF \l_@@_rules_color_tl
6459 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6460 \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`.

```

6461 \seq_map_inline:Nn \g_@@_cols_vlism_seq
6462 {
6463   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6464   {
6465     \int_compare:nNnT
6466       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6467     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

6468       \@@_qpoint:n { col - ##1 }
6469       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6470       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6471       \pgfusepathqstroke
6472     }
6473   }
6474 }

```

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.

```

6475 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6476 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6477 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6478 {
6479   \bool_lazy_and:nnTF
6480     { \int_compare_p:nNn { ##1 } > 0 }
6481     {
6482       \int_compare_p:nNn
6483         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6484     {
6485       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6486       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6487       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6488       \pgfusepathqstroke
6489     }
6490     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
6491 }

```

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.

```

6492 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6493 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6494 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
6495 {
6496   \bool_lazy_and:nnTF
6497     { \int_compare_p:nNn { ##1 } > 0 }
6498     {
6499       \int_compare_p:nNn
6500         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6501     {
6502       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

6503 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

6504 \dim_set:Nn \l_tmpa_dim
6505 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6506 \str_case:nn { #1 }
6507 {
6508   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6509   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
6510   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6511   }
6512   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```


We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

6513         \dim_set:Nn \l_tmpb_dim
6514         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6515         \str_case:nn { #2 }
6516         {
6517             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6518             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
6519             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6520         }
6521         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6522         \pgfusepathqstroke
6523         \group_end:
6524     }
6525     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
6526 }

```

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).

```

6527     \str_if_empty:NF \l_@@_submatrix_name_str
6528     {
6529         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
6530         \l_@@_x_initial_dim \l_@@_y_initial_dim
6531         \l_@@_x_final_dim \l_@@_y_final_dim
6532     }
6533     \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`.

```

6534     \begin { pgfscope }
6535     \pgftransformshift
6536     {
6537         \pgfpoint
6538         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6539         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6540     }
6541     \str_if_empty:NTF \l_@@_submatrix_name_str
6542     { \@@_node_left:nn #1 { } }
6543     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
6544     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

6545     \pgftransformshift
6546     {
6547         \pgfpoint
6548         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6549         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6550     }
6551     \str_if_empty:NTF \l_@@_submatrix_name_str
6552     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
6553     {
6554         \@@_node_right:nnnn #2
6555         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
6556     }
6557     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
6558     \flag_clear_new:n { nicematrix }
6559     \l_@@_code_tl
6560 }

```

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`.

```
6561 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpntanchor
```

The following command will be linked to `\pgfpntanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpntanchor` and we apply to it the command `\@@_pgfpntanchor_i:nn` before passing it to the original `\pgfpntanchor`. We have to act in an expandable way because the command `\pgfpntanchor` is used in names of Tikz nodes which are computed in an expandable way.

```
6562 \cs_new_protected:Npn \@@_pgfpntanchor:n #1
6563 {
6564   \use:e
6565   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpntanchor_i:nn #1 } }
6566 }
```

In fact, the argument of `\pgfpntanchor` 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 -.

```
6567 \cs_new:Npn \@@_pgfpntanchor_i:nn #1 #2
6568 { #1 { \@@_pgfpntanchor_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.

```
6569 \tl_const:Nn \c_@@_integers_alist_tl
6570 {
6571   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
6572   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
6573   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
6574   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
6575 }

6576 \cs_new:Npn \@@_pgfpntanchor_ii:w #1-#2\q_stop
6577 {
```

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 `\pgfpntanchor` and, the, the j arrives (alone) in the following `\pgfpntanchor`. In order to know whether we have a number of row of a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```
6578   \tl_if_empty:nTF { #2 }
6579   {
6580     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
6581     {
6582       \flag_raise:n { nicematrix }
6583       \int_if_even:nTF { \flag_height:n { nicematrix } }
6584       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
6585       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
6586     }
6587     { #1 }
6588   }
```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```
6589   { \@@_pgfpntanchor_iii:w { #1 } #2 }
6590 }
```

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. `\@@_pgfpntanchor_i:nn`).

```
6591 \cs_new:Npn \@@_pgfpntanchor_iii:w #1 #2 -
```

```

6592 {
6593   \str_case:nnF { #1 }
6594   {
6595     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
6596     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
6597   }

```

Now the case of a node of the form $i-j$.

```

6598 {
6599   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
6600   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
6601 }
6602 }

```

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`).

```

6603 \cs_new_protected:Npn \@@_node_left:nn #1 #2
6604 {
6605   \pgfnode
6606   { rectangle }
6607   { east }
6608   {
6609     \nullfont
6610     \c_math_toggle_token
6611     \tl_if_empty:NF \l_@@_delimiters_color_tl
6612     { \color { \l_@@_delimiters_color_tl } }
6613     \left #1
6614     \vcenter
6615     {
6616       \nullfont
6617       \hrule \@height \l_tmpa_dim
6618       \@depth \c_zero_dim
6619       \@width \c_zero_dim
6620     }
6621     \right .
6622     \c_math_toggle_token
6623   }
6624   { #2 }
6625   { }
6626 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument `#3` is the subscript and `#4` is the superscript.

```

6627 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
6628 {
6629   \pgfnode
6630   { rectangle }
6631   { west }
6632   {
6633     \nullfont
6634     \c_math_toggle_token
6635     \tl_if_empty:NF \l_@@_delimiters_color_tl
6636     { \color { \l_@@_delimiters_color_tl } }
6637     \left .
6638     \vcenter
6639     {
6640       \nullfont
6641       \hrule \@height \l_tmpa_dim
6642       \@depth \c_zero_dim
6643       \@width \c_zero_dim
6644     }

```

```

6645     \right #1
6646     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
6647     ^ { \smash { #4 } }
6648     \c_math_toggle_token
6649   }
6650   { #2 }
6651   { }
6652 }

```

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.

```

6653 \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.

```

6654 \bool_new:N \c_@@_footnote_bool

6655 \@@_msg_new:nnn { Unknown-key-for-package }
6656 {
6657   The-key~'\l_keys_key_str'~is-unknown. \\
6658   If-you-go-on,~it~will~be~ignored. \\
6659   For-a-list-of~the~available~keys,~type-H~<return>.
6660 }
6661 {
6662   The-available-keys-are~(in~alphabetic~order):~
6663   footnote,~
6664   footnotehyper,~
6665   renew-dots,~and
6666   renew-matrix.
6667 }

```

Maybe we will completely delete the key 'transparent' in a future version.

```

6668 \@@_msg_new:nn { Key-transparent }
6669 {
6670   The-key~'transparent'~is-now-obsolete~(because~it's~name~
6671   is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
6672   and~'renew-matrix'.~However,~you~can~go~on.
6673 }

6674 \keys_define:nn { NiceMatrix / Package }
6675 {
6676   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
6677   renew-dots .value_forbidden:n = true ,
6678   renew-matrix .code:n = \@@_renew_matrix: ,
6679   renew-matrix .value_forbidden:n = true ,
6680   transparent .code:n =
6681   {
6682     \@@_renew_matrix:
6683     \bool_set_true:N \l_@@_renew_dots_bool
6684     \@@_error:n { Key-transparent }
6685   } ,
6686   transparent .value_forbidden:n = true,
6687   footnote .bool_set:N = \c_@@_footnote_bool ,
6688   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
6689   unknown .code:n = \@@_error:n { Unknown-key-for-package }

```

```

6690 }
6691 \ProcessKeysOptions { NiceMatrix / Package }

6692 \@@_msg_new:nn { footnote~with~footnotehyper~package }
6693 {
6694   You~can't~use~the~option~'footnote'~because~the~package~
6695   footnotehyper~has~already~been~loaded.~
6696   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
6697   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6698   of~the~package~footnotehyper.\\
6699   If~you~go~on,~the~package~footnote~won't~be~loaded.
6700 }

6701 \@@_msg_new:nn { footnotehyper~with~footnote~package }
6702 {
6703   You~can't~use~the~option~'footnotehyper'~because~the~package~
6704   footnote~has~already~been~loaded.~
6705   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
6706   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6707   of~the~package~footnote.\\
6708   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
6709 }

6710 \bool_if:NT \c_@@_footnote_bool
6711 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6712   \@ifclassloaded { beamer }
6713   { \bool_set_false:N \c_@@_footnote_bool }
6714   {
6715     \@ifpackageloaded { footnotehyper }
6716     { \@_error:n { footnote~with~footnotehyper~package } }
6717     { \usepackage { footnote } }
6718   }
6719 }

6720 \bool_if:NT \c_@@_footnotehyper_bool
6721 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6722   \@ifclassloaded { beamer }
6723   { \bool_set_false:N \c_@@_footnote_bool }
6724   {
6725     \@ifpackageloaded { footnote }
6726     { \@_error:n { footnotehyper~with~footnote~package } }
6727     { \usepackage { footnotehyper } }
6728   }
6729   \bool_set_true:N \c_@@_footnote_bool
6730 }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

Error messages of the package

The following message will be deleted when we will delete the key `except-corners` for the command `\arraycolor`.

```

6731 \@@_msg_new:nn { key except-corners }
6732 {
6733   The~key~'except-corners'~has~been~deleted~for~the~command~\token_to_str:N

```

```

6734 \arraycolor\ in-the-\token_to_str:N \CodeBefore.~You~should~instead~use~
6735 the~key~'corners'~in~your~\@@_full_name_env:.\
6736 If~you~go~on,~this~key~will~be~ignored.
6737 }

6738 \seq_new:N \c_@@_types_of_matrix_seq
6739 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
6740 {
6741   NiceMatrix ,
6742   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
6743 }
6744 \seq_set_map_x:NNn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
6745 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is executed. This command raises an error but try to give the best information to the user in the error message. The command `\seq_if_in:NVTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

6746 \cs_new_protected:Npn \@@_error_too_much_cols:
6747 {
6748   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
6749   {
6750     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
6751     { \@@_fatal:n { too~much~cols~for~matrix } }
6752     {
6753       \bool_if:NF \l_@@_last_col_without_value_bool
6754       { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
6755     }
6756   }
6757   { \@@_fatal:n { too~much~cols~for~array } }
6758 }

```

The following command must *not* be protected since it's used in an error message.

```

6759 \cs_new:Npn \@@_message_hdotsfor:
6760 {
6761   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
6762   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
6763 }

6764 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
6765 {
6766   You~try~to~use~more~columns~than~allowed~by~your~
6767   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
6768   columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
6769   exterior~columns).~This~error~is~fatal.
6770 }

6771 \@@_msg_new:nn { too~much~cols~for~matrix }
6772 {
6773   You~try~to~use~more~columns~than~allowed~by~your~
6774   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
6775   number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
6776   'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
6777   This~error~is~fatal.
6778 }

```

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`.

```

6779 \@@_msg_new:nn { too~much~cols~for~array }
6780 {
6781   You~try~to~use~more~columns~than~allowed~by~your~
6782   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
6783   \int_use:N \g_@@_static_num_of_col_int\
6784   ~(plus~the~potential~exterior~ones).~

```

```

6785     This-error-is-fatal.
6786 }
6787 \@@_msg_new:nn { last-col-not-used }
6788 {
6789     The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
6790     in~your~\@@_full_name_env:~.~However,~you~can~go~on.
6791 }
6792 \@@_msg_new:nn { columns-not-used }
6793 {
6794     The-preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
6795     \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\
6796     You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
6797 }
6798 \@@_msg_new:nn { in-first-col }
6799 {
6800     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\
6801     If~you~go~on,~this~command~will~be~ignored.
6802 }
6803 \@@_msg_new:nn { in-last-col }
6804 {
6805     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\
6806     If~you~go~on,~this~command~will~be~ignored.
6807 }
6808 \@@_msg_new:nn { in-first-row }
6809 {
6810     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\
6811     If~you~go~on,~this~command~will~be~ignored.
6812 }
6813 \@@_msg_new:nn { in-last-row }
6814 {
6815     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\
6816     If~you~go~on,~this~command~will~be~ignored.
6817 }
6818 \@@_msg_new:nn { double-closing-delimiter }
6819 {
6820     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
6821     delimiter.~This~delimiter~will~be~ignored.
6822 }
6823 \@@_msg_new:nn { delimiter-after-opening }
6824 {
6825     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
6826     delimiter.~This~delimiter~will~be~ignored.
6827 }
6828 \@@_msg_new:nn { bad-option-for-line-style }
6829 {
6830     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
6831     is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
6832 }
6833 \@@_msg_new:nn { Unknown-key-for-xdots }
6834 {
6835     As~for~now,~there~is~only~three~keys~available~here:~'color',~'line-style'~
6836     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
6837     this~key~will~be~ignored.
6838 }
6839 \@@_msg_new:nn { Unknown-key-for-RowStyle }
6840 {
6841     As~for~now,~there~is~only~three~keys~available~here:~'cell-space-top-limit',~
6842     'cell-space-bottom-limit~and~'cell-space-limits'~(and~you~try~to~use~
6843     '\l_keys_key_str').~If~you~go~on,~this~key~will~be~ignored.
6844 }

```

```

6845 \@@_msg_new:nn { Unknown-key-for-rowcolors }
6846 {
6847   As-for-now,~there-is-only~two-keys-available-here:~'cols'~and~'respect-blocks'~
6848   (and-you-try-to-use~'\l_keys_key_str').~If-you-go-on,~
6849   this-key-will-be-ignored.
6850 }
6851 \@@_msg_new:nn { ampersand-in-light-syntax }
6852 {
6853   You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
6854   ~you-have-used-the-key~'light-syntax'.~This-error-is-fatal.
6855 }
6856 \@@_msg_new:nn { SubMatrix-too-large }
6857 {
6858   Your-command~\token_to_str:N \SubMatrix\
6859   can't-be-drawn-because-your-matrix-is-too-small.\\
6860   If-you-go-on,~this-command-will-be-ignored.
6861 }
6862 \@@_msg_new:nn { double-backslash-in-light-syntax }
6863 {
6864   You-can't-use~\token_to_str:N \\~to-separate-rows-because-you-have-used~
6865   the-key~'light-syntax'.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
6866   (set-by-the-key~'end-of-row').~This-error-is-fatal.
6867 }
6868 \@@_msg_new:nn { standard-cline-in-document }
6869 {
6870   The-key~'standard-cline'~is-available-only-in-the-preamble.\\
6871   If-you-go-on~this-command-will-be-ignored.
6872 }
6873 \@@_msg_new:nn { bad-value-for-baseline }
6874 {
6875   The-value-given-to~'baseline'~(\int_use:N \l_tmpa_int)~is-not~
6876   valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
6877   \int_use:N \g_@@_row_total_int\ or-equal-to~'t',~'c'~or~'b'.\\
6878   If-you-go-on,~a-value-of~1~will-be-used.
6879 }
6880 \@@_msg_new:nn { Invalid-name-format }
6881 {
6882   You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
6883   \SubMatrix.\\
6884   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
6885   If-you-go-on,~this-key-will-be-ignored.
6886 }
6887 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
6888 {
6889   You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
6890   \token_to_str:N \SubMatrix\ of-your~\@@_full_name_env:\ but-that~
6891   number-is-not-valid.~If-you-go-on,~it-will-be-ignored.
6892 }
6893 \@@_msg_new:nn { impossible-delimiter }
6894 {
6895   It's-impossible-to-draw-the~#1~delimiter~of-your~
6896   \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
6897   in-that-column.
6898   \bool_if:NT \l_@@_submatrix_slim_bool
6899   { ~Maybe-you-should-try-without-the-key~'slim'. } \\
6900   If-you-go-on,~this~\token_to_str:N \SubMatrix\ will-be-ignored.
6901 }
6902 \@@_msg_new:nn { width-without-X-columns }
6903 {
6904   You-have-used-the-key~'width'~but-you-have-put-no~'X'~column. \\

```



```

6905     If~you~go~on,~that~key~will~be~ignored.
6906 }

6907 \@@_msg_new:nn { empty~environment }
6908 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }

6909 \@@_msg_new:nn { Delimiter~with~small }
6910 {
6911     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
6912     because~the~key~'small'~is~in~force.\\
6913     This~error~is~fatal.
6914 }

6915 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
6916 {
6917     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code~after'~
6918     can't~be~executed~because~a~cell~doesn't~exist.\\
6919     If~you~go~on~this~command~will~be~ignored.
6920 }

6921 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
6922 {
6923     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
6924     in~this~\@@_full_name_env:.\
6925     If~you~go~on,~this~key~will~be~ignored.\\
6926     For~a~list~of~the~names~already~used,~type~H~<return>.
6927 }
6928 {
6929     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
6930     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
6931 }

6932 \@@_msg_new:nn { r~or~l~with~preamble }
6933 {
6934     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
6935     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
6936     your~\@@_full_name_env:.\
6937     If~you~go~on,~this~key~will~be~ignored.
6938 }

6939 \@@_msg_new:nn { Hdotsfor~in~col~0 }
6940 {
6941     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
6942     the~array.~This~error~is~fatal.
6943 }

6944 \@@_msg_new:nn { bad~corner }
6945 {
6946     #1~is~an~incorrect~specification~for~a~corner~(in~the~keys~
6947     'corners'~and~'except~corners').~The~available~
6948     values~are:~NW,~SW,~NE~and~SE.\\
6949     If~you~go~on,~this~specification~of~corner~will~be~ignored.
6950 }

6951 \@@_msg_new:nn { bad~border }
6952 {
6953     #1~is~an~incorrect~specification~for~a~border~(in~the~key~
6954     'borders'~of~the~command~\token_to_str:N \Block).~The~available~
6955     values~are:~left,~right,~top~and~bottom.\\
6956     If~you~go~on,~this~specification~of~border~will~be~ignored.
6957 }

6958 \@@_msg_new:nn { tikz~key~without~tikz }
6959 {
6960     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
6961     \Block'~because~you~have~not~loaded~Tikz.~
6962     If~you~go~on,~this~key~will~be~ignored.
6963 }

```

```

6964 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
6965 {
6966   In-the-\@@_full_name_env:,~you~must~use~the~key~
6967   'last-col'~without~value.\\
6968   However,~you~can~go~on~for~this~time~
6969   (the~value~'\l_keys_value_tl'~will~be~ignored).
6970 }
6971 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
6972 {
6973   In~\NiceMatrixoptions,~you~must~use~the~key~
6974   'last-col'~without~value.\\
6975   However,~you~can~go~on~for~this~time~
6976   (the~value~'\l_keys_value_tl'~will~be~ignored).
6977 }
6978 \@@_msg_new:nn { Block-too-large-1 }
6979 {
6980   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
6981   too~small~for~that~block. \\
6982 }
6983 \@@_msg_new:nn { Block-too-large-2 }
6984 {
6985   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
6986   \g_@@_static_num_of_col_int\
6987   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
6988   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
6989   (&)~at~the~end~of~the~first~row~of~your~
6990   \@@_full_name_env:.\
6991   If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
6992 }
6993 \@@_msg_new:nn { unknown-column-type }
6994 {
6995   The~column~type~'#1'~in~your~\@@_full_name_env:\
6996   is~unknown. \\
6997   This~error~is~fatal.
6998 }
6999 \@@_msg_new:nn { tabularnote-forbidden }
7000 {
7001   You~can't~use~the~command~\token_to_str:N\tabularnote\
7002   ~in~a~\@@_full_name_env:~This~command~is~available~only~in~
7003   \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
7004   If~you~go~on,~this~command~will~be~ignored.
7005 }
7006 \@@_msg_new:nn { borders-forbidden }
7007 {
7008   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
7009   because~the~option~'rounded-corners'~
7010   is~in~force~with~a~non-zero~value.\\
7011   If~you~go~on,~this~key~will~be~ignored.
7012 }
7013 \@@_msg_new:nn { bottomrule-without-booktabs }
7014 {
7015   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
7016   loaded~'booktabs'.\\
7017   If~you~go~on,~this~key~will~be~ignored.
7018 }
7019 \@@_msg_new:nn { enumitem-not-loaded }
7020 {
7021   You~can't~use~the~command~\token_to_str:N\tabularnote\
7022   ~because~you~haven't~loaded~'enumitem'.\\
7023   If~you~go~on,~this~command~will~be~ignored.
7024 }

```

```

7025 \@@_msg_new:nn { Wrong-last-row }
7026 {
7027   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
7028   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
7029   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
7030   last-row.~You~can~avoid~this~problem~by~using~'last-row'~
7031   without~value~(more~compilations~might~be~necessary).
7032 }
7033 \@@_msg_new:nn { Yet-in-env }
7034 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }
7035 \@@_msg_new:nn { Outside-math-mode }
7036 {
7037   The~\@@_full_name_env:\ can~be~used~only~in~math-mode~
7038   (and~not~in~\token_to_str:N \vcenter).\\
7039   This~error~is~fatal.
7040 }
7041 \@@_msg_new:nn { One-letter-allowed }
7042 {
7043   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
7044   If~you~go~on,~it~will~be~ignored.
7045 }
7046 \@@_msg_new:nnn { Unknown-key-for-Block }
7047 {
7048   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
7049   \Block.\\ If~you~go~on,~it~will~be~ignored. \\
7050   For~a~list~of~the~available~keys,~type~H~<return>.
7051 }
7052 {
7053   The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
7054   hvlines,~l,~line-width,~rounded-corners,~r,~t~and~tikz.
7055 }
7056 \@@_msg_new:nnn { Unknown-key-for-CodeAfter }
7057 {
7058   The~key~'\l_keys_key_str'~is~unknown.\\
7059   If~you~go~on,~it~will~be~ignored. \\
7060   For~a~list~of~the~available~keys~in~\token_to_str:N
7061   \CodeAfter,~type~H~<return>.
7062 }
7063 {
7064   The~available~keys~are~(in~alphabetic~order):~
7065   delimiters/color,~
7066   rules~(with~the~subkeys~'color'~and~'width'),~
7067   sub-matrix~(several~subkeys)~
7068   and~xdots~(several~subkeys).~
7069   The~latter~is~for~the~command~\token_to_str:N \line.
7070 }
7071 \@@_msg_new:nnn { Unknown-key-for-SubMatrix }
7072 {
7073   The~key~'\l_keys_key_str'~is~unknown.\\
7074   If~you~go~on,~this~key~will~be~ignored. \\
7075   For~a~list~of~the~available~keys~in~\token_to_str:N
7076   \SubMatrix,~type~H~<return>.
7077 }
7078 {
7079   The~available~keys~are~(in~alphabetic~order):~
7080   'delimiters/color',~
7081   'extra-height',~
7082   'hlines',~
7083   'hvlines',~
7084   'left-xshift',~
7085   'name',~

```

```

7086     'right-xshift',~
7087     'rules'~(with~the~subkeys~'color'~and~'width'),~
7088     'slim',~
7089     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
7090     and~'right-xshift').\\
7091 }
7092 \@@_msg_new:nnn { Unknown~key~for~notes }
7093 {
7094     The~key~'\l_keys_key_str'~is~unknown.\\
7095     If~you~go~on,~it~will~be~ignored. \\
7096     For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
7097 }
7098 {
7099     The~available~keys~are~(in~alphabetic~order):~
7100     bottomrule,~
7101     code-after,~
7102     code-before,~
7103     enumitem-keys,~
7104     enumitem-keys-para,~
7105     para,~
7106     label-in-list,~
7107     label-in-tabular~and~
7108     style.
7109 }
7110 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
7111 {
7112     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7113     \token_to_str:N \NiceMatrixOptions. \\
7114     If~you~go~on,~it~will~be~ignored. \\
7115     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7116 }
7117 {
7118     The~available~keys~are~(in~alphabetic~order):~
7119     allow-duplicate-names,~
7120     cell-space-bottom-limit,~
7121     cell-space-limits,~
7122     cell-space-top-limit,~
7123     code-for-first-col,~
7124     code-for-first-row,~
7125     code-for-last-col,~
7126     code-for-last-row,~
7127     corners,~
7128     create-extra-nodes,~
7129     create-medium-nodes,~
7130     create-large-nodes,~
7131     delimiters~(several~subkeys),~
7132     end-of-row,~
7133     first-col,~
7134     first-row,~
7135     hlines,~
7136     hvlines,~
7137     last-col,~
7138     last-row,~
7139     left-margin,~
7140     letter-for-dotted-lines,~
7141     light-syntax,~
7142     notes~(several~subkeys),~
7143     nullify-dots,~
7144     renew-dots,~
7145     renew-matrix,~
7146     right-margin,~
7147     rules~(with~the~subkeys~'color'~and~'width'),~
7148     small,~

```

```

7149     sub-matrix~(several~subkeys),
7150     vlines,~
7151     xdots~(several~subkeys).
7152 }

7153 \@@_msg_new:nnn { Unknown-key~for~NiceArray }
7154 {
7155     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7156     \{NiceArray\}. \\
7157     If~you~go~on,~it~will~be~ignored. \\
7158     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7159 }
7160 {
7161     The~available~keys~are~(in~alphabetic~order):~
7162     b,~
7163     baseline,~
7164     c,~
7165     cell-space-bottom-limit,~
7166     cell-space-limits,~
7167     cell-space-top-limit,~
7168     code-after,~
7169     code-for-first-col,~
7170     code-for-first-row,~
7171     code-for-last-col,~
7172     code-for-last-row,~
7173     colortbl-like,~
7174     columns-width,~
7175     corners,~
7176     create-extra-nodes,~
7177     create-medium-nodes,~
7178     create-large-nodes,~
7179     delimiters/color,~
7180     extra-left-margin,~
7181     extra-right-margin,~
7182     first-col,~
7183     first-row,~
7184     hlines,~
7185     hvlines,~
7186     last-col,~
7187     last-row,~
7188     left-margin,~
7189     light-syntax,~
7190     name,~
7191     notes/bottomrule,~
7192     notes/para,~
7193     nullify-dots,~
7194     renew-dots,~
7195     right-margin,~
7196     rules~(with~the~subkeys~'color'~and~'width'),~
7197     small,~
7198     t,~
7199     tabularnote,~
7200     vlines,~
7201     xdots/color,~
7202     xdots/shorten~and~
7203     xdots/line-style.
7204 }

```

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).

```

7205 \@@_msg_new:nnn { Unknown-key~for~NiceMatrix }
7206 {
7207     The~key~'\l_keys_key_str'~is~unknown~for~the~
7208     \@@_full_name_env:. \\

```

```

7209     If~you~go~on,~it~will~be~ignored.  \
7210     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7211 }
7212 {
7213     The~available~keys~are~(in~alphabetic~order):~
7214     b,~
7215     baseline,~
7216     c,~
7217     cell-space-bottom-limit,~
7218     cell-space-limits,~
7219     cell-space-top-limit,~
7220     code-after,~
7221     code-for-first-col,~
7222     code-for-first-row,~
7223     code-for-last-col,~
7224     code-for-last-row,~
7225     colortbl-like,~
7226     columns-width,~
7227     corners,~
7228     create-extra-nodes,~
7229     create-medium-nodes,~
7230     create-large-nodes,~
7231     delimiters~(several~subkeys),~
7232     extra-left-margin,~
7233     extra-right-margin,~
7234     first-col,~
7235     first-row,~
7236     hlines,~
7237     hvlines,~
7238     l,~
7239     last-col,~
7240     last-row,~
7241     left-margin,~
7242     light-syntax,~
7243     name,~
7244     nullify-dots,~
7245     r,~
7246     renew-dots,~
7247     right-margin,~
7248     rules~(with~the~subkeys~'color'~and~'width'),~
7249     small,~
7250     t,~
7251     vlines,~
7252     xdots/color,~
7253     xdots/shorten~and~
7254     xdots/line-style.
7255 }
7256 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
7257 {
7258     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7259     \{NiceTabular\}.  \
7260     If~you~go~on,~it~will~be~ignored.  \
7261     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7262 }
7263 {
7264     The~available~keys~are~(in~alphabetic~order):~
7265     b,~
7266     baseline,~
7267     c,~
7268     cell-space-bottom-limit,~
7269     cell-space-limits,~
7270     cell-space-top-limit,~
7271     code-after,~

```

```

7272 code-for-first-col,~
7273 code-for-first-row,~
7274 code-for-last-col,~
7275 code-for-last-row,~
7276 colortbl-like,~
7277 columns-width,~
7278 corners,~
7279 create-extra-nodes,~
7280 create-medium-nodes,~
7281 create-large-nodes,~
7282 extra-left-margin,~
7283 extra-right-margin,~
7284 first-col,~
7285 first-row,~
7286 hlines,~
7287 hvlines,~
7288 last-col,~
7289 last-row,~
7290 left-margin,~
7291 light-syntax,~
7292 name,~
7293 notes/bottomrule,~
7294 notes/para,~
7295 nullify-dots,~
7296 renew-dots,~
7297 right-margin,~
7298 rules~(with~the~subkeys~'color'~and~'width'),~
7299 t,~
7300 tabularnote,~
7301 vlines,~
7302 xdots/color,~
7303 xdots/shorten~and~
7304 xdots/line-style.
7305 }

7306 \@@_msg_new:nnn { Duplicate-name }
7307 {
7308   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
7309   the~same~environment~name~twice.~You~can~go~on,~but,~
7310   maybe,~you~will~have~incorrect~results~especially~
7311   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
7312   message~again,~use~the~key~'allow-duplicate-names'~in~
7313   '\token_to_str:N \NiceMatrixOptions'.\\
7314   For~a~list~of~the~names~already~used,~type~H~<return>. \\
7315 }
7316 {
7317   The~names~already~defined~in~this~document~are:~
7318   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
7319 }

7320 \@@_msg_new:nn { Option-auto-for-columns-width }
7321 {
7322   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
7323   If~you~go~on,~the~key~will~be~ignored.
7324 }

```

19 History

The successive versions of the file `nicematrix.sty` provided by TeXLive are available on the svn server of TeXLive:

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

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types L, C and R.

Changes between version 1.2 and 1.3

New environment `{pNiceArrayC}` and its variants.
Correction of a bug in the definition of `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` (in fact, it was a typo).
Options are now available locally in `{pNiceMatrix}` and its variants.
The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types `w` and `W` can now be used in the environments `{NiceArray}`, `{pNiceArrayC}` and its variants with the same meaning as in the package `array`.
New option `columns-width` to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of `nicematrix` were focused on the continuous dotted lines whereas the version 2.0 of `nicematrix` provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment `{pNiceArrayRC}`. With this new implementation, there is no restriction on the width of the columns.
The package `nicematrix` no longer loads `mathtools` but only `amsmath`.
Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option `code-for-first-row` is now available in the command `\NiceMatrixOptions`.
Following a discussion on TeX StackExchange⁶⁵, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁶⁶

⁶⁵cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁶⁶Before 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 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`).

Possibility to draw vertical dotted lines to separate columns with the specifier “:” in the preamble (similar to the classical specifier “|” and the specifier “:” of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “:” in the preamble.

Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.

Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of “|”) as `\hdotsfor` does.

Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.

Error message when the user gives an incorrect value for `last-row`.

A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol “:” (in the preamble of the array) and `\line` in `code-after`).

The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.

The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by `colortbl`.

Correction of a bug: it was not possible to use the colon “:” in the preamble of an array when `pdflatex` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

The option `columns-width=auto` doesn't need any more a second compilation.

The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).

The previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange⁶⁷, optimization of Tikz externalization is disabled in the environments of `nicematrix` when the class `standalone` or the package `standalone` is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

Changes between version 3.5 and 3.6

LaTeX counters `iRow` and `jCol` available in the cells of the array.

Addition of `\normalbaselines` before the construction of the array: in environments like `{align}` of `amsmath` the value of `\baselineskip` is changed and if the options `first-row` and `last-row` were used in an environment of `nicematrix`, the position of the delimiters was wrong.

A warning is written in the `.log` file if an obsolete environment is used.

There is no longer artificial errors `Duplicate-name` in the environments of `amsmath`.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.

New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

⁶⁷cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

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](https://stackoverflow.com)).
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 `hylvline` don't draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It's now possible to write `\begin{pNiceMatrix}a&b\\c&d\end{pNiceMatrix}`² with the expected result.

Changes between versions 4.2 and 4.3

The horizontal centering of the content of a `\Block` is correct even when an instruction such as `!\qqquad` is used in the preamble of the array.

It's now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.

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

Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).

Environment `{NiceTabular*}`

Command `\Vdotsfor` similar to `\Hdotsfor`

The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.

Key `respect-blocks` for `\rowcolors` (with a *s*) in the `code-before`.

The variable `\g_nicematrix_code_before_tl` is now public.

The key `baseline` may take in as value an expression of the form *line-i* to align the `\hline` in the row *i*.

The key `hvlines-except-corners` may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys `c`, `r` and `l` for the command `\Block`.

It's possible to use the key `draw-first` with `\Ddots` and `\Iddots` to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key `tabularnote`.

Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put `\omit` before `\CodeAfter`.

Correction of a bug: the tabular notes `\tabularnotes` were not composed when present in a block (except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.

New command `\NotEmpty`.

Changes between versions 5.6 and 5.7

New key `delimiters-color`

Keys `fill`, `draw` and `line-width` for the command `\Block`.

Changes between versions 5.7 and 5.8

Keys `cols` and `restart` of the command `\rowcolors` in the `code-before`.

Modification of the behaviour of `\\` in the columns of type `p`, `m` or `b` (for a behaviour similar to the environments of `array`).

Better error messages for the command `\Block`.

Changes between versions 5.8 and 5.9

Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.

New key `cell-space-limits`.

Changes between versions 5.9 and 5.10

New command `\SubMatrix` available in the `\CodeAfter`.

It's possible to provide options (between brackets) to the keyword `\CodeAfter`.

A (non fatal) error is raised when the key `transparent`, which is deprecated, is used.

Changes between versions 5.10 and 5.11

It's now possible, in the `code-before` and in the `\CodeAfter`, to use the syntax `|(i-|j)` for the Tikz node at the intersection of the (potential) horizontal rule number i and the (potential) vertical rule number j .

Changes between versions 5.11 and 5.12

Keywords `\CodeBefore` and `\Body` (alternative syntax to the key `code-before`).

New key `delimiters/max-width`.

New keys `hlines`, `vlines` and `hvlines` for the command `\SubMatrix` in the `\CodeAfter`.

New key `rounded-corners` for the command `\Block`.

Changes between versions 5.12 and 5.13

New command `\arraycolor` in the `\CodeBefore` (with its key `except-corners`).

New key `borders` for the command `\Block`.

New command `\Hline` (for horizontal rules not drawn in the blocks).

The keys `vlines` and `hlines` takes in as value a (comma-separated) list of numbers (for the rules to draw).

Changes between versions 5.13 and 5.14

Nodes of the form `(1.5)`, `(2.5)`, `(3.5)`, etc.

Keys `t` and `b` for the command `\Block`.

Key `corners`.

Changes between versions 5.14 and 5.15

Key `hvlines` for the command `\Block`.

The commands provided by `nicematrix` to color cells, rows and columns don't color the cells which are in the "corners" (when the key `corner` is used).

It's now possible to specify delimiters for submatrices in the preamble of an environment.

The version 5.15b is compatible with the version 3.0+ of `siunitx` (previous versions were not).

Changes between versions 5.15 and 5.16

It's now possible to use the cells corresponding to the contents of the nodes (of the form `i-j`) in the `\CodeBefore` when the key `create-cell-nodes` of that `\CodeBefore` is used. The medium and the large nodes are also available if the corresponding keys are used.

Changes between versions 5.16 and 5.17

The key `define-L-C-R` (only available at load-time) now raises a (non fatal) error.

Keys `L`, `C` and `R` for the command `\Block`.

Key `hvlines-except-borders`.

It's now possible to use a key `l`, `r` or `c` with the command `\pAutoNiceMatrix` (and the similar ones).

Changes between versions 5.17 and 5.18

New command `\RowStyle`

Changes between versions 5.18 and 5.19

New key `tikz` for the command `\Block`.

Changes between versions 5.19 and 6.0

Columns X and environment {NiceTabularX}.

Command \rowlistcolors available in the \CodeBefore.

In columns with fixed width, the blocks are composed as paragraphs (wrapping of the lines).

The key define-L-C-R has been deleted.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
@@ commands:	
\@@_Block:	1237, 5350
\@@_Block_i	5355, 5356, 5360
\@@_Block_ii:nnnnn	5360, 5361
\@@_Block_iv:nnnnn	5398, 5402
\@@_Block_iv:nnnnnn	5614, 5616
\@@_Block_v:nnnnn	5399, 5521
\@@_Block_v:nnnnnn	5643, 5646
\@@_Cdots	1161, 1228, 3899
\g_@@_Cdots_lines_tl	1258, 3041
\@@_Cell:	198, 217, 880, 1792, 1901, 1949, 2076, 2186, 2207, 2228, 2841
\@@_CodeAfter:	1241, 6212
\@@_CodeAfter_i:n	883, 2719, 2764, 6212, 6213, 6223
\@@_CodeAfter_ii:n	6216, 6218
\@@_CodeAfter_keys:	2981, 3002
\@@_CodeBefore:w	1432, 1434
\@@_CodeBefore_keys:	1412, 1429
\@@_Ddots	1163, 1230, 3931
\g_@@_Ddots_lines_tl	1261, 3039
\g_@@_HVDotsfor_lines_tl	1263, 3037, 4015, 4092, 6761
\@@_Hdotsfor:	1166, 1235, 3991
\@@_Hdotsfor:nnnn	4017, 4030
\@@_Hdotsfor_i	4000, 4006, 4013
\@@_Hline:	1233, 4844
\@@_Hline_i:n	4844, 4845, 4851
\@@_Hline_ii:nn	4848, 4851
\@@_Hline_iii:n	4849, 4852
\@@_Hspace:	1234, 3985
\@@_Iddots	1164, 1231, 3955
\g_@@_Iddots_lines_tl	1262, 3040
\@@_Ldots	1160, 1165, 1227, 3883
\g_@@_Ldots_lines_tl	1259, 3042
\l_@@_Matrix_bool	273, 1642, 1666, 2325, 2663, 2667, 2675, 2832
\l_@@_NiceArray_bool	270, 404, 1303, 1518, 1582, 1705, 1712, 1724, 2325, 2651, 2663, 2667, 2675, 2808, 4697, 4701, 4830, 4834, 5051, 5058, 6130
\g_@@_NiceMatrixBlock_int	257, 5110, 5115, 5118, 5129
\l_@@_NiceTabular_bool	168, 271, 888, 1087, 1411, 1414, 1485, 1615, 1619, 1713, 1725, 2652, 2863, 2873, 2882, 2980, 2983, 5431, 5530, 6138
\@@_NotEmpty:	1243, 2855
\@@_OnlyMainNiceMatrix:n	1239, 4554
\@@_OnlyMainNiceMatrix_i:n	4557, 4564, 4567
\@@_RowStyle:n	1244, 1355
\@@_SubMatrix	2962, 6362
\@@_SubMatrix_in_code_before	1404, 6342
\@@_Vdots	1162, 1229, 3915
\g_@@_Vdots_lines_tl	1260, 3038
\@@_Vdotsfor:	1236, 4090
\@@_Vdotsfor:nnnn	4094, 4105
\@@_W:	1669, 1752, 2176
\l_@@_X_column_bool	275, 2077, 5396
\l_@@_X_columns_aux_bool	302, 1542, 1553, 2070
\l_@@_X_columns_dim	303, 1543, 1554, 2071
\@@_actually_color:	1413, 4231
\@@_actually_diagbox:nnnnnn	5409, 5712, 6150, 6166
\@@_actually_draw_Cdots:	3455, 3459
\@@_actually_draw_Ddots:	3605, 3609
\@@_actually_draw_Iddots:	3657, 3661
\@@_actually_draw_Ldots:	3416, 3420, 4081
\@@_actually_draw_Vdots:	3537, 3541, 4156
\@@_adapt_S_column:	229, 234, 247, 1484
\@@_add_to_colors_seq:nn	4217, 4229, 4230, 4259, 4277, 4286, 4393
\@@_adjust_pos_of_blocks_seq:	2949, 3004
\@@_adjust_pos_of_blocks_seq_i:nnnn	3007, 3009
\@@_adjust_size_box:	961, 988, 1958, 2237, 2743, 2788
\@@_adjust_to_submatrix:nn	3300, 3403, 3442, 3523, 3598, 3650
\@@_adjust_to_submatrix:nnnnnn	3307, 3309
\@@_after_array:	1651, 2886
\g_@@_after_col_zero_bool	304, 1128, 2720, 3997
\@@_analyze_end:Nn	2473, 2518
\l_@@_argspec_tl	3881, 3882, 3883, 3899, 3915, 3931, 3955, 4011, 4012, 4013, 4088, 4089, 4090, 4166, 4167, 4168, 6360, 6361, 6362
\@@_array:	1082, 2474, 2501
\@@_arraycolor	1401, 4323
\c_@@_arydshln_loaded_bool	24, 31, 1778
\l_@@_auto_columns_width_bool	507, 653, 2585, 2589, 5105
\g_@@_aux_found_bool	1275, 1280, 1436, 1506, 1509
\g_@@_aux_tl	276, 1512, 1551, 1658, 2895, 2909, 2917, 2990, 4927

<code>\l_@@_bar_at_end_of_pream_bool</code>	1706, 1836, 2655	2904, 3066, 3490, 3508, 3568, 4151, 4544, 5156, 5166, 5200, 5287, 5626, 5837, 6384, 6433
<code>\l_@@_baseline_tl</code> 498, 499, 646, 647, 648, 649, 1095, 1584, 2278, 2290, 2295, 2297, 2302, 2307, 2397, 2398, 2402, 2407, 2409, 2414		<code>\l_@@_col_width_dim</code>
<code>\@@_begin_of_NiceMatrix:nn</code> 2830, 2851		259, 260, 1900, 1947, 5435, 5438
<code>\@@_begin_of_row:</code> 886, 910, 1187, 2721		<code>\@@_color_index:n</code>
<code>\l_@@_block_auto_columns_width_bool</code> ..		4383, 4404, 4407
..... 1498, 2590, 5098, 5103, 5113, 5123		<code>\l_@@_color_int</code>
<code>\g_@@_block_box_int</code> 348, 1478, 5404, 5418, 5420, 5474, 5486, 5498, 5507, 5517		4347, 4348, 4365, 4366, 4386, 4397
<code>\@@_block_tikz:nnnnn</code> 5700, 6040		<code>\l_@@_color_tl</code>
<code>\g_@@_blocks_dp_dim</code>		339, 4380, 4381, 4391, 4394, 5347, 5422, 5424
..... 267, 969, 972, 973, 5501, 5504		<code>\g_@@_colors_seq</code> 1408, 4220, 4224, 4225, 4235
<code>\g_@@_blocks_ht_dim</code>		<code>\l_@@_colors_seq</code> 4342, 4343, 4387, 4406, 4408
..... 266, 975, 978, 979, 5492, 5495		<code>\@@_colortbl_like:</code>
<code>\g_@@_blocks_seq</code>		1168, 1245
..... 319, 1500, 2335, 5511, 5523, 5614		<code>\l_@@_colortbl_like_bool</code> 484, 669, 1245, 1694
<code>\g_@@_blocks_wd_dim</code>		<code>\c_@@_colortbl_loaded_bool</code> . 100, 104, 1195
..... 265, 963, 966, 967, 5480, 5483		<code>\l_@@_cols_tl</code>
<code>\c_@@_booktabs_loaded_bool</code> 25, 34, 1176, 2368		4252, 4300, 4334, 4344, 4345, 4395, 4442, 4445
<code>\l_@@_borders_clist</code> 337, 5584, 5673, 5982, 5998, 6000, 6002, 6004, 6023, 6035		<code>\g_@@_cols_vlism_seq</code> .. 284, 1689, 1769, 6461
<code>\@@_cartesian_color:nn</code> 4249, 4261, 4270, 4395		<code>\@@_columncolor</code>
<code>\@@_cartesian_path:</code> 4253, 4494		1402, 4264
<code>\@@_cartesian_path:n</code> 4301, 4436, 4494		<code>\@@_columncolor_preamble</code>
<code>\l_@@_cell_box</code> 887, 935, 937, 943, 949, 952, 956, 965, 966, 971, 972, 977, 978, 989, 990, 991, 992, 994, 997, 1001, 1003, 1022, 1037, 1039, 1046, 1047, 1060, 1178, 1288, 1290, 1926, 1930, 1934, 1937, 1948, 1959, 2079, 2227, 2238, 2722, 2746, 2749, 2751, 2768, 2791, 2795, 5720, 5830, 5867, 6145		1172, 4542
<code>\l_@@_cell_space_bottom_limit_dim</code> ..		<code>\c_@@_columncolor_regex</code>
..... 487, 560, 992, 1344		59, 1697
<code>\l_@@_cell_space_top_limit_dim</code>		<code>\l_@@_columns_width_dim</code>
..... 486, 558, 990, 1335		258, 654, 796, 2586, 2592, 5111, 5117
<code>\@@_cellcolor</code> 1395, 4303, 4315, 4316		<code>\g_@@_com_or_env_str</code>
<code>\@@_cellcolor_tabular</code> 1170, 4519		288, 291
<code>\g_@@_cells_seq</code> 2512, 2513, 2514, 2516		<code>\@@_computations_for_large_nodes:</code> ...
<code>\@@_center_cell_box:</code> 1890, 1921	 5227, 5240, 5245
<code>\@@_chessboardcolors</code> 1403, 4308		<code>\@@_computations_for_medium_nodes:</code> ..
<code>\@@_cline</code> 147, 1226	 5147, 5216, 5226, 5237
<code>\@@_cline_i:nn</code> 148, 149, 161, 164		<code>\@@_compute_a_corner:nnnnnn</code>
<code>\@@_cline_i:w</code> 149, 150	 4915, 4917, 4919, 4921, 4934
<code>\l_@@_code_before_bool</code> 308, 643, 670, 1102, 1294, 1325, 1515, 2532, 2549, 2567, 2598, 2624, 2658, 2695, 2995, 2997		<code>\@@_compute_corners:</code>
<code>\g_@@_code_before_tl</code> . 1504, 1513, 1516, 2992		2948, 4907
<code>\l_@@_code_before_tl</code>		<code>\@@_construct_preamble:</code>
..... 307, 642, 1324, 1412, 1516		1300, 1663
<code>\l_@@_code_for_first_col_tl</code> 577, 2733		<code>\l_@@_corners_cells_seq</code>
<code>\l_@@_code_for_first_row_tl</code> . 581, 898, 5799		323, 2954, 4439, 4479, 4628, 4634, 4641, 4763, 4769, 4776, 4909, 4925, 4929, 4930, 4990
<code>\l_@@_code_for_last_col_tl</code> 579, 2777		<code>\l_@@_corners_clist</code>
<code>\l_@@_code_for_last_row_tl</code> . 583, 905, 5802	 503, 631, 636, 4594, 4730, 4910
<code>\l_@@_code_tl</code> 296, 6337, 6559		<code>\@@_create_col_nodes:</code> 2477, 2505, 2524
<code>\l_@@_col_max_int</code>		<code>\@@_create_diag_nodes:</code> ... 1384, 2925, 3108
..... 331, 3167, 3178, 3246, 3305, 3322		<code>\@@_create_extra_nodes:</code>
<code>\l_@@_col_min_int</code> 1467, 2334, 3100, 5137
..... 330, 3172, 3235, 3240, 3303, 3320		<code>\@@_create_large_nodes:</code>
<code>\g_@@_col_total_int</code> 262, 1005, 1254, 1367, 1378, 1449, 1568, 2157, 2158, 2617, 2618, 2698, 2702, 2707, 2708, 2767, 2890, 2892,		5145, 5221
		<code>\@@_create_medium_and_large_nodes:</code> ..
	 5142, 5232
		<code>\@@_create_medium_nodes:</code>
		5143, 5211
		<code>\@@_create_nodes:</code> 5218, 5229, 5239, 5242, 5283
		<code>\@@_create_row_node:</code> 1098, 1131, 1177
		<code>\@@_cut_on_hyphen:w</code>
	 4244, 4293, 4298, 4361, 4449, 4450, 4472, 4473, 4503, 4504, 5900, 5909, 5940, 5943, 5987, 5990, 6346, 6351, 6376, 6379
		<code>\g_@@_ddots_int</code>
		2928, 3629, 3630
		<code>\@@_def_env:nnn</code>
	 2814, 2825, 2826, 2827, 2828, 2829
		<code>\@@_define_com:nnn</code>
	 6114, 6122, 6123, 6124, 6125, 6126
		<code>\@@_delimiter:nnn</code>
		... 1992, 2013, 2021, 2034, 2040, 2046, 6226
		<code>\l_@@_delimiters_color_tl</code>
		. 520, 729, 1425, 1607, 1608, 1625, 1626, 6206, 6264, 6265, 6292, 6611, 6612, 6635, 6636
		<code>\l_@@_delimiters_max_width_bool</code>
	 521, 727, 1630
		<code>\g_@@_delta_x_one_dim</code> 2930, 3632, 3642

\g_@@_delta_x_two_dim	2932, 3680, 3690	5789, 5807, 5844, 6241, 6244, 6400, 6402,
\g_@@_delta_y_one_dim	2931, 3634, 3642	6407, 6409, 6436, 6438, 6443, 6445, 6543, 6555
\g_@@_delta_y_two_dim	2933, 3682, 3690	\g_@@_env_int
\@@_diagbox:nn	1242, 6146	. 251, 252, 254, 1497, 1507, 1510, 1657, 5326
\@@_dotfill:	6135	\@@_error:n
\@@_dotfill_i:	6140, 6142	12,
\@@_dotfill_ii:	6139, 6142, 6143	380, 405, 530, 540, 593, 723, 779, 789, 795,
\@@_dotfill_iii:	6143, 6144	804, 812, 830, 837, 845, 846, 847, 853,
\@@_double_int_eval:n	4162, 4176, 4177	858, 859, 860, 874, 876, 877, 878, 1353,
\g_@@_dp_ante_last_row_dim	914, 1211	1427, 1538, 1563, 1573, 1645, 2312, 2373,
\g_@@_dp_last_row_dim		2419, 4322, 4337, 5574, 5609, 5980, 6210,
. . . . 914, 915, 1214, 1215, 1289, 1290, 1601		6322, 6333, 6340, 6385, 6684, 6689, 6716, 6726
\g_@@_dp_row_zero_dim		\@@_error:nn
. . . . 934, 935, 1205, 1206, 1594, 2391, 2430		13, 661,
\@@_draw_Cdots:nnn	3440	1995, 2041, 2047, 3886, 3889, 3902, 3905,
\@@_draw_Ddots:nnn	3596	3918, 3921, 3935, 3936, 3941, 3942, 3959,
\@@_draw_Iddots:nnn	3648	3960, 3965, 3966, 4923, 5985, 6327, 6415, 6418
\@@_draw_Ldots:nnn	3401	\@@_error:nnn
\@@_draw_Vdots:nnn	3521	14, 4193, 6490, 6525
\@@_draw_blocks:	2335, 5611	\@@_error_too_much_cols:
\@@_draw_dotted_lines:	2947, 3026	1734, 6746
\@@_draw_dotted_lines_i:	3029, 3033	\@@_everycr:
\l_@@_draw_first_bool . 346, 3946, 3970, 3981		1124, 1200, 1203
\@@_draw_hlines:	2960, 4826	\@@_everycr_i:
\@@_draw_line:	3438,	1124, 1125
3483, 3594, 3646, 3694, 3696, 4215, 5066, 5096		\l_@@_except_borders_bool
\@@_draw_line_ii:nn	4195, 4199 512, 605, 4697, 4701, 4830, 4834
\@@_draw_line_iii:nn	4202, 4206	\@@_exec_code_before:
\@@_draw_non_standard_dotted_line: . .		1294, 1406
. 3702, 3704		\@@_expand_clist:NN
\@@_draw_non_standard_dotted_line:n . .		4442, 4443, 4495
. 3707, 3710		\l_@@_exterior_arraycolsep_bool
\@@_draw_non_standard_dotted_line:nnn	 500, 792, 1715, 1727, 2654
. 3712, 3717, 3731		\l_@@_extra_left_margin_dim
\@@_draw_standard_dotted_line: . 3701, 3732	 515, 621, 1316, 2754
\@@_draw_standard_dotted_line_i: 3795, 3799		\l_@@_extra_right_margin_dim
\l_@@_draw_tl 335, 5578,	 516, 622, 1533, 2799, 3571
5582, 5661, 5882, 5888, 5890, 5892, 5929, 5930		\@@_extract_brackets
\@@_draw_vlines: 2961, 4693		5685, 5872
\g_@@_empty_cell_bool . . 316, 996, 1006,		\@@_extract_coords_values: 5308, 5315
2756, 2803, 3897, 3913, 3929, 3953, 3976, 3987		\@@_fatal:n 15, 280, 1488, 1973, 2002,
\@@_end_Cell: 203, 219, 982,		2482, 2486, 2488, 2521, 4002, 6751, 6754, 6757
1794, 1917, 1954, 2085, 2188, 2217, 2233, 2841		\@@_fatal:nn 16, 1783, 2180
\l_@@_end_of_row_tl		\l_@@_fill_tl 334, 5576, 5683, 5685
. 517, 518, 571, 2497, 2498, 6865		\l_@@_final_i_int
\c_@@_endpgfortikzpicture_tl 2937, 3154, 3159, 3162, 3187,
. 43, 47, 3030, 4203, 5037		3195, 3199, 3208, 3216, 3296, 3351, 3432,
\c_@@_enumitem_loaded_bool		3505, 3511, 3514, 4035, 4063, 4131, 4141, 4143
. 26, 37, 377, 697, 702, 713, 718		\l_@@_final_j_int
\@@_env: 252, 256, 920, 926, 1023,		2938, 3155, 3160, 3167, 3172, 3178, 3188,
1029, 1043, 1051, 1107, 1113, 1119, 1190,		3196, 3200, 3209, 3217, 3297, 3352, 3385,
1368, 1369, 1374, 1375, 1380, 1381, 1392,		3388, 3396, 3622, 4056, 4066, 4068, 4110, 4139
1446, 1447, 1452, 1455, 1458, 1461, 2533,		\l_@@_final_open_bool 2940, 3161,
2536, 2538, 2554, 2560, 2563, 2572, 2578,		3165, 3168, 3175, 3181, 3185, 3201, 3429,
2581, 2603, 2609, 2612, 2629, 2635, 2641,		3464, 3469, 3480, 3544, 3554, 3559, 3580,
2665, 2673, 2687, 2698, 2702, 2708, 2971,		3619, 3669, 3803, 3818, 3849, 3850, 4033,
3068, 3072, 3078, 3085, 3091, 3125, 3127,		4057, 4069, 4108, 4132, 4144, 4185, 5034, 5071
3134, 3136, 3137, 3142, 3145, 3207, 3275,		\@@_find_extremities_of_line:nnnn . . .
3339, 3350, 3363, 3366, 3385, 3388, 3493,	 3149, 3406, 3445, 3526, 3601, 3653
3496, 3511, 3514, 4044, 4062, 4119, 4137,		\l_@@_first_col_int
4188, 4190, 4209, 4212, 4945, 4964, 4982,	 135, 148, 358, 359, 573, 851, 886,
5169, 5171, 5179, 5290, 5299, 5317, 5735,		1378, 1449, 1577, 1707, 2527, 2547, 2902,
5744, 5748, 5762, 5767, 5779, 5785, 5786,		3066, 3490, 3508, 3995, 4463, 4556, 5156,
		5166, 5200, 5248, 5287, 6095, 6101, 6107, 6433
		\l_@@_first_i_tl . 6372, 6377, 6396, 6427,
		6436, 6438, 6493, 6500, 6502, 6584, 6595, 6599
		\l_@@_first_j_tl 6373, 6378, 6400,
		6402, 6463, 6476, 6483, 6485, 6585, 6596, 6600
		\l_@@_first_row_int
	 356, 357, 574, 855, 1252,
		1372, 1444, 1592, 2309, 2388, 2416, 2427,

2899, 3064, 3360, 3382, 5149, 5163, 5190,
 5247, 5285, 5741, 5759, 6093, 6238, 6397, 6876
 \c_@@_footnote_bool
 1473, 1661, 6654, 6687, 6710, 6713, 6723, 6729
 \c_@@_footnotehyper_bool . 6653, 6688, 6720
 \@@_full_name_env:
 289, 6735, 6767, 6774, 6782, 6790, 6794,
 6890, 6908, 6911, 6924, 6929, 6934, 6936,
 6966, 6985, 6990, 6995, 7002, 7028, 7037, 7208
 \@@_hdottedline: 1232, 5019
 \@@_hdottedline:n 5027, 5031
 \@@_hdottedline:i: 5022, 5024
 \@@_hdottedline:i:n 5036, 5040
 \@@_hline:nn 4711, 4841, 4860
 \@@_hline:i:nn 2958, 4714, 4717
 \@@_hline_i_complete:nn 2958, 4824
 \@@_hline_ii:nnnn ... 4739, 4750, 4783, 4825
 \l_@@_hlines_clist 352, 585, 599,
 604, 633, 1132, 1134, 1138, 2960, 4839, 4840
 \l_@@_hpos_block_str 341, 342, 5331,
 5333, 5335, 5337, 5339, 5341, 5376, 5377,
 5379, 5430, 5439, 5452, 5463, 5514, 5537,
 5541, 5553, 5558, 5590, 5592, 5594, 5596,
 5599, 5602, 5811, 5823, 5834, 5838, 5848, 5860
 \l_@@_hpos_cell_str 263, 264,
 1792, 1881, 1882, 1950, 2186, 2229, 5375, 5377
 \l_@@_hpos_col_str
 1839, 1841, 1843, 1868, 1880, 1882, 1883, 2063
 \l_@@_hpos_of_block_cap_bool
 343, 5597, 5600, 5603, 5738, 5808, 5845
 \g_@@_ht_last_row_dim
 916, 1212, 1213, 1287, 1288, 1600
 \g_@@_ht_row_one_dim .. 942, 943, 1209, 1210
 \g_@@_ht_row_zero_dim
 936, 937, 1207, 1208, 1595, 2390, 2429
 \@@_hvlines_block:nnn 5655, 5936
 \l_@@_hvlines_block_bool .. 347, 5586, 5651
 \@@_i: 5149, 5151,
 5152, 5153, 5154, 5163, 5169, 5171, 5172,
 5173, 5174, 5179, 5180, 5181, 5182, 5190,
 5193, 5195, 5196, 5197, 5249, 5251, 5254,
 5255, 5259, 5260, 5285, 5290, 5292, 5294,
 5298, 5299, 5310, 5317, 5319, 5321, 5325, 5326
 \g_@@_iddots_int 2929, 3677, 3678
 \l_@@_in_env_bool 269, 404, 1488, 1489
 \c_@@_in_preamble_bool . 21, 22, 23, 693, 709
 \l_@@_initial_i_int 2935,
 3152, 3227, 3230, 3255, 3263, 3267, 3276,
 3284, 3294, 3340, 3425, 3471, 3473, 3487,
 3493, 3496, 4034, 4035, 4045, 4113, 4123, 4125
 \l_@@_initial_j_int
 2936, 3153, 3228, 3235,
 3240, 3246, 3256, 3264, 3268, 3277, 3285,
 3295, 3341, 3363, 3366, 3374, 3561, 3563,
 3568, 3614, 4038, 4048, 4050, 4109, 4110, 4121
 \l_@@_initial_open_bool
 2939, 3229, 3233, 3236, 3243, 3249,
 3253, 3269, 3422, 3461, 3468, 3478, 3544,
 3551, 3557, 3611, 3663, 3801, 3848, 4032,
 4039, 4051, 4107, 4114, 4126, 4184, 5033, 5070
 \@@_insert_tabularnotes: 2340, 2343
 \@@_instruction_of_type:nnn
 1063, 3891, 3907, 3923, 3946, 3970
 \c_@@_integers_alist_tl 6569, 6580
 \l_@@_inter_dots_dim
 . 488, 489, 2944, 3806, 3813, 3824, 3832,
 3839, 3844, 3856, 3864, 5062, 5064, 5092, 5094
 \g_@@_internal_code_after_tl 297, 1828,
 1991, 2012, 2020, 2033, 2039, 2045, 2096,
 2976, 2977, 4859, 5026, 5407, 5710, 6148, 6354
 \@@_intersect_our_row:nnnn 4425
 \@@_intersect_our_row_p:nnnn 4375
 \@@_j: 5156, 5158,
 5159, 5160, 5161, 5166, 5169, 5171, 5174,
 5176, 5177, 5179, 5182, 5184, 5185, 5200,
 5203, 5205, 5206, 5207, 5262, 5264, 5267,
 5269, 5273, 5274, 5287, 5290, 5291, 5293,
 5298, 5299, 5311, 5317, 5318, 5320, 5325, 5326
 \l_@@_l_dim
 3779, 3780, 3793, 3794, 3806, 3812,
 3823, 3831, 3839, 3844, 3856, 3857, 3864, 3865
 \l_@@_large_nodes_bool 511, 612, 5141, 5145
 \g_@@_last_col_found_bool 366,
 1257, 1569, 1637, 2616, 2690, 2765, 2889, 5835
 \l_@@_last_col_int
 364, 365, 780, 823, 825, 838, 854,
 875, 1278, 1281, 1572, 1719, 2837, 2839,
 2890, 2892, 3531, 3566, 3888, 3904, 3942,
 3966, 5619, 5624, 5625, 5626, 5629, 5658,
 5668, 5680, 5692, 5704, 5716, 5731, 5762,
 5767, 5775, 5791, 6097, 6103, 6109, 6750, 6768
 \l_@@_last_col_without_value_bool ...
 363, 822, 2891, 6753
 \l_@@_last_empty_column_int
 4955, 4956, 4969, 4975, 4988
 \l_@@_last_empty_row_int
 4937, 4938, 4951, 4972
 \l_@@_last_i_tl 6374,
 6380, 6383, 6396, 6430, 6443, 6445, 6493, 6500
 \l_@@_last_j_tl
 6375, 6381, 6384, 6407, 6409, 6466, 6476, 6483
 \l_@@_last_row_int
 360, 361, 575, 903, 950, 1144, 1272,
 1276, 1283, 1557, 1561, 1564, 1576, 1598,
 2499, 2500, 2729, 2730, 2774, 2775, 2894,
 3411, 3450, 3920, 3936, 3960, 4562, 4570,
 5618, 5621, 5622, 5641, 5658, 5668, 5680,
 5692, 5703, 5715, 5729, 5790, 5801, 6105, 7027
 \l_@@_last_row_without_value_bool ...
 362, 1274, 1559, 2893
 \l_@@_left_delim_dim
 1301, 1305, 1310, 2465, 2752
 \g_@@_left_delim_tl
 1309, 1475, 1609, 1633, 1703, 1976, 1978, 5061
 \l_@@_left_margin_dim
 513, 615, 1315, 2753, 5052, 5278
 \l_@@_letter_for_dotted_lines_str ...
 803, 814, 815, 1764
 \l_@@_letter_vlism_tl 283, 592, 1767
 \l_@@_light_syntax_bool 497, 569, 1318, 1528
 \@@_light_syntax_i 2490, 2493
 \@@_line 2975, 4168
 \@@_line_i:nn 4175, 4182
 \l_@@_line_width_dim
 340, 5588, 5883, 5921, 5932, 5938, 5950,
 5977, 5997, 6012, 6014, 6024, 6025, 6027, 6038

<code>\@@_line_with_light_syntax:n</code> ...	2504, 2508	<code>\l_@@_nullify_dots_bool</code>	
<code>\@@_line_with_light_syntax_i:n</code>	506, 610, 3895, 3911, 3927, 3951, 3974
.....	2503, 2509, 2510	<code>\l_@@_number_of_notes_int</code>	370, 407, 417, 427
<code>\@@_math_toggle.token:</code>		<code>\@@_old_CT@arc@</code>	1490, 3000
.....	167, 984, 2723, 2740, 2769, 2785, 6190, 6194	<code>\@@_old_cdots</code>	1220, 3912
<code>\g_@@_max_cell_width_dim</code>		<code>\@@_old_ddots</code>	1222, 3952
.....	993, 994, 1499, 2591, 5104, 5130	<code>\@@_old_dotfill</code>	6134, 6137, 6145
<code>\c_@@_max_l_dim</code>	3793, 3798	<code>\@@_old_dotfill:</code>	1240
<code>\l_@@_medium_nodes_bool</code>	510, 611, 5139, 5782	<code>\l_@@_old_iRow_int</code>	298, 1268, 3046
<code>\@@_message_hdotsfor:</code>	6759, 6767, 6774, 6782	<code>\@@_old_ialign:</code>	1097, 1216, 5613
<code>\@@_msg_new:nn</code>	17,	<code>\@@_old_iddots</code>	1223, 3975
.....	6668, 6692, 6701, 6731, 6764, 6771, 6779,	<code>\l_@@_old_jCol_int</code>	299, 1270, 3047
.....	6787, 6792, 6798, 6803, 6808, 6813, 6818,	<code>\@@_old_ldots</code>	1219, 3896
.....	6823, 6828, 6833, 6839, 6845, 6851, 6856,	<code>\@@_old_multicolumn</code>	1249, 3990
.....	6862, 6868, 6873, 6880, 6887, 6893, 6902,	<code>\@@_old_pgfpaintanchor</code>	175, 6561, 6565
.....	6907, 6909, 6915, 6932, 6939, 6944, 6951,	<code>\@@_old_pgful@check@rerun</code>	93, 97
.....	6958, 6964, 6971, 6978, 6983, 6993, 6999,	<code>\@@_old_vdots</code>	1221, 3928
.....	7006, 7013, 7019, 7025, 7033, 7035, 7041, 7320	<code>\@@_open_x_final_dim:</code>	
<code>\@@_msg_new:nnn</code> ...	18, 6655, 6921, 7046,	3379, 3431, 3465, 3623, 3672
.....	7056, 7071, 7092, 7110, 7153, 7205, 7256, 7306	<code>\@@_open_x_initial_dim:</code>	
<code>\@@_msg_redirect_name:nn</code>	3357, 3424, 3462, 3616, 3666
.....	19, 798, 1649, 5632, 5635	<code>\@@_open_y_final_dim:</code>	3503, 3555, 3621, 3671
<code>\@@_multicolumn:nnn</code>	1247, 2126	<code>\@@_open_y_initial_dim:</code>	
<code>\g_@@_multicolumn_cells_seq</code>	3485, 3552, 3613, 3665
.....	326, 1250, 1296, 2141,	<code>\l_@@_parallelize_diags_bool</code>	
.....	2915, 2919, 2920, 5174, 5182, 5304, 5746, 5764	501, 502, 607, 2926, 3627, 3675
<code>\g_@@_multicolumn_sizes_seq</code>		<code>\@@_patch_m_preamble:n</code>	
.....	327, 1251, 1297, 2143, 2921, 2922, 5305	2135, 2161, 2195, 2200, 2266
<code>\g_@@_name_env_str</code>	287, 292,	<code>\@@_patch_m_preamble_i:n</code>	
.....	293, 1482, 1483, 2520, 2809, 2810, 2818,	2165, 2166, 2167, 2182
.....	2819, 2848, 2861, 2869, 2879, 2998, 6118, 6748	<code>\@@_patch_m_preamble_ii:nn</code>	
<code>\l_@@_name_str</code>	509, 663,	2168, 2169, 2170, 2192
.....	922, 925, 1025, 1028, 1115, 1118, 2537,	<code>\@@_patch_m_preamble_iii:n</code>	2171, 2197
.....	2538, 2562, 2563, 2580, 2581, 2611, 2612,	<code>\@@_patch_m_preamble_iv:nnn</code>	
.....	2637, 2640, 2683, 2686, 2704, 2707, 3126,	2172, 2173, 2174, 2202
.....	3127, 3138, 3141, 3144, 5295, 5298, 5322, 5325	<code>\@@_patch_m_preamble_ix:n</code>	2251, 2269
<code>\g_@@_names_seq</code>	268, 660, 662, 7318	<code>\@@_patch_m_preamble_v:nnnn</code>	2175, 2176, 2222
<code>\l_@@_nb_cols_int</code>		<code>\@@_patch_m_preamble_vi:n</code>	2177, 2243
.....	6071, 6082, 6090, 6096, 6102, 6108	<code>\@@_patch_m_preamble_x:n</code>	
<code>\l_@@_nb_rows_int</code>	6070, 6081, 6099	2190, 2220, 2241, 2246, 2248, 2272
<code>\@@_newcolumn_type</code>	1151, 1668, 1669	<code>\@@_patch_node_for_cell:</code>	1035, 1417
<code>\@@_node_for_cell:</code>		<code>\@@_patch_node_for_cell:n</code>	1033, 1059, 1062
.....	1002, 1009, 1417, 2750, 2800	<code>\@@_patch_preamble:n</code>	
<code>\@@_node_for_multicolumn:nn</code>	5306, 5313	1691, 1737, 1773, 1781, 1802, 1833,
<code>\@@_node_left:nn</code>	6542, 6543, 6603	1980, 1996, 1998, 2014, 2022, 2048, 2098, 2118
<code>\@@_node_position:</code>		<code>\@@_patch_preamble_i:n</code>	1741, 1742, 1743, 1788
.....	1374, 1376, 1380, 1382, 1446, 1448, 1456, 1462	<code>\@@_patch_preamble_ii:nn</code>	
<code>\@@_node_position_i:</code>	1459, 1463	1744, 1745, 1746, 1799
<code>\@@_node_right:nnnn</code>	6552, 6554, 6627	<code>\@@_patch_preamble_iii:n</code> .	1747, 1804, 1812
<code>\g_@@_not_empty_cell_bool</code>		<code>\@@_patch_preamble_iii_i:n</code>	1807, 1809
.....	306, 1000, 1007, 2856	<code>\@@_patch_preamble_iv:n</code>	
<code>\@@_not_in_exterior:nnnn</code>	4417	1748, 1749, 1750, 1853
<code>\@@_not_in_exterior_p:nnnn</code>	4353	<code>\@@_patch_preamble_iv_i:n</code>	1856, 1858
<code>\l_@@_notes_above_space_dim</code>	504, 505	<code>\@@_patch_preamble_iv_ii:w</code>	1861, 1862, 1864
<code>\l_@@_notes_bottomrule_bool</code>		<code>\@@_patch_preamble_iv_iii:nn</code> ...	1865, 1866
.....	681, 841, 869, 2366	<code>\@@_patch_preamble_iv_iv:n</code>	1870, 1872, 2071
<code>\l_@@_notes_code_after_tl</code>	679, 2375	<code>\@@_patch_preamble_iv_v:nnnn</code> ...	1876, 1895
<code>\l_@@_notes_code_before_tl</code>	677, 2347	<code>\@@_patch_preamble_ix:n</code>	1760, 2051
<code>\@@_notes_label_in_list:n</code>	373, 392, 400, 689	<code>\@@_patch_preamble_ix_i:w</code>	2054, 2055, 2057
<code>\@@_notes_label_in_tabular:n</code> .	372, 413, 686	<code>\@@_patch_preamble_ix_ii:n</code>	2058, 2061
<code>\l_@@_notes_para_bool</code> ..	675, 839, 867, 2351	<code>\@@_patch_preamble_v:nnnn</code>	1751, 1752, 1942
<code>\@@_notes_style:n</code>		<code>\@@_patch_preamble_vi:n</code>	1753, 1965
.....	371, 374, 392, 400, 416, 421, 683		

\@@_patch_preamble_vii:nn	\l_@@_real_left_delim_dim 2436, 2451, 2466
..... 1754, 1755, 1756, 1971	\l_@@_real_right_delim_dim 2437, 2463, 2469
\@@_patch_preamble_vii_i:nn 1984, 1987, 1989	\@@_recreate_cell_nodes: 1385, 1442
\@@_patch_preamble_viii:nn	\g_@@_recreate_cell_nodes_bool
..... 1757, 1758, 1759, 2000 508, 1185, 1385, 1409, 1416, 1421
\@@_patch_preamble_viii_i:nnn .. 2004, 2026	\@@_rectanglecolor .. 1396, 4273, 4306, 4326
\@@_patch_preamble_x:n	\@@_rectanglecolor:nnn ... 4279, 4288, 4291
... 1797, 1893, 1963, 1969, 2089, 2100, 2124	\@@_renew_NC@rewrite@S: 186, 190, 210, 1256
\@@_patch_preamble_xi:n	\@@_renew_dots: 1158, 1246
\@@_patch_preamble_xii:n	\l_@@_renew_dots_bool
\@@_pgf_rect_node:nnn 608, 788, 1246, 6676, 6683
..... 460, 1460, 5784	\@@_renew_matrix: 783, 787, 6050, 6678, 6682
\@@_pgf_rect_node:nnnnn	\l_@@_respect_blocks_bool 4332, 4349, 4372
..... 435, 5289, 5316, 5734, 5778, 6529	\@@_restore_iRow_jCol: 2999, 3044
\c_@@_pgfortikzpicture_tl	\@@_revtex_array: 1074, 1085
..... 42, 46, 3028, 4201, 5035	\c_@@_revtex_bool 50, 52, 55, 57, 1084
\@@_pgfpointanchor:n	\l_@@_right_delim_dim
..... 6557, 6562 1302, 1306, 1312, 2468, 2797
\@@_pgfpointanchor_i:nn	\g_@@_right_delim_tl .. 1311, 1476, 1627,
..... 6565, 6567	1633, 1704, 1979, 2008, 2009, 2030, 2035, 5063
\@@_pgfpointanchor_ii:w	\l_@@_right_margin_dim
..... 6568, 6576 514, 617, 1532, 2798, 3570, 5059, 5281
\@@_pgfpointanchor_iii:w	\@@_rotate: 1238, 4161
..... 6589, 6591	\g_@@_rotate_bool
\@@_picture_position: 274, 959, 987, 1957, 2236, 2742,
..... 1369, 1376, 1382, 1448, 1462, 1463	2787, 4161, 5430, 5471, 5476, 5536, 5552, 5721
\g_@@_pos_of_blocks_seq	\@@_rotate_cell_box:
320, 1295, 1501, 2144, 2907, 2911, 2912, 947, 987, 1957, 2236, 2742, 2787, 5721
2952, 3006, 4351, 4588, 4724, 5002, 5648, 6158	\l_@@_rounded_corners_dim
\g_@@_pos_of_stroken_blocks_seq	338, 5580, 5693, 5897, 5898, 5933, 5979, 6036
..... 322, 1502, 4592, 4728, 5670	\@@_roundedrectanglecolor 1397, 4282
\g_@@_pos_of_xdots_seq	\l_@@_row_max_int 329, 3162, 3304, 3321
..... 321, 1503, 2953, 3292, 4590, 4726	\l_@@_row_min_int 328, 3230, 3302, 3319
\g_@@_post_action_cell_tl	\g_@@_row_of_col_done_bool
..... 882, 986, 1334, 1343, 1923, 2078 305, 1129, 1481, 2546
\@@_pre_array: 1266, 1326, 1525	\g_@@_row_style_tl
\@@_pre_array_i:w	893, 912, 1332, 1341, 1357, 1505, 1909, 5425
..... 1322, 1525	\g_@@_row_total_int 261, 1253, 1366,
\@@_pre_array_ii: 1174, 1298	1372, 1444, 1575, 2310, 2417, 2894, 2901,
\@@_pre_code_before: 1362, 1438	3064, 3360, 3382, 4076, 5149, 5163, 5190,
\c_@@_preamble_first_col_tl 1708, 2715	5285, 5641, 5741, 5759, 6238, 6383, 6397, 6877
\c_@@_preamble_last_col_tl 1720, 2760	\@@_rowcolor 1398, 4255
\g_@@_preamble_tl	\@@_rowcolor_tabular 1171, 4530
.... 1477, 1670, 1674, 1678, 1684, 1699,	\@@_rowcolors 1399, 4410
1708, 1717, 1720, 1729, 1733, 1771, 1780,	\@@_rowcolors_i:nnnn 4376, 4412
1790, 1801, 1814, 1897, 1944, 1967, 1983,	\l_@@_rowcolors_restart_bool ... 4335, 4364
2011, 2019, 2032, 2038, 2073, 2094, 2107,	\@@_rowlistcolors 1400, 4339, 4411
2114, 2123, 2134, 2136, 2184, 2194, 2199,	\g_@@_rows_seq . 2496, 2498, 2500, 2502, 2504
2204, 2224, 2245, 2255, 2262, 2271, 2474, 2501	\l_@@_rows_tl
\@@_pred:n 4251, 4299, 4378, 4395, 4443, 4468
.... 136, 166, 2839, 4629, 4642, 4764, 4777	\l_@@_rules_color_tl
\@@_provide_pgfsyspdfmark: ... 60, 69, 1472 300, 544, 1522, 1523, 6458, 6459
\@@_put_box_in_flow: 1635, 2274, 2467	\@@_set_CT@arc@: 169, 1523, 6459
\@@_put_box_in_flow_bis:nn 1632, 2434	\@@_set_CT@arc@_i: 170, 171
\@@_put_box_in_flow_i: 2280, 2282	\@@_set_CT@arc@_ii: 170, 173
\@@_qpoint:n 255, 2285, 2287, 2299,	\@@_set_final_coords: 3330, 3355
2315, 2382, 2384, 2400, 2411, 2422, 3114,	\@@_set_final_coords_from_anchor:n ..
3116, 3118, 3120, 3130, 3132, 3374, 3396,	... 3346, 3435, 3466, 3547, 3556, 3626, 3674
3425, 3432, 3471, 3473, 3487, 3505, 3561,	\@@_set_initial_coords: 3325, 3344
3563, 3614, 3622, 4209, 4212, 4462, 4466,	\@@_set_initial_coords_from_anchor:n .
4482, 4484, 4652, 4654, 4656, 4787, 4789,	... 3335, 3428, 3463, 3546, 3553, 3618, 3668
4791, 5043, 5047, 5054, 5088, 5091, 5093,	\@@_set_size:n 6068, 6083
5195, 5205, 5725, 5727, 5729, 5731, 5755,	\c_@@_siunitx_loaded_bool 176, 180, 185, 231
5775, 5804, 5905, 5907, 5914, 5916, 5951,	
5953, 5957, 5962, 5964, 5968, 6011, 6013,	
6015, 6022, 6026, 6028, 6171, 6173, 6176,	
6178, 6231, 6233, 6427, 6430, 6468, 6485, 6502	
\l_@@_radius_dim 492, 493, 2095,	
2943, 3436, 3437, 3873, 5021, 5045, 5089, 5090	

<code>\c_@@_size_seq</code>	<code>\@@_test_in_corner_v:</code>
... 1276, 1281, 1364, 1365, 1366, 1367, 2897	4595, 4623
<code>\l_@@_small_bool</code>	<code>\@@_test_vline_in_block:nnnn</code>
781, 828, 834,	4589, 4591, 4874
856, 891, 1180, 1973, 2002, 2724, 2770, 2941	
<code>\@@_standard_cline</code>	<code>\@@_test_vline_in_stroken_block:nnnn</code>
132, 1225	4593, 4896
<code>\@@_standard_cline:w</code>	
132, 133	<code>\l_@@_the_array_box</code>
<code>\l_@@_standard_cline_bool</code> ..	1299, 1314, 1547, 2327, 2328, 2330, 2333
485, 556, 1224	
<code>\c_@@_standard_tl</code> 495, 496, 3700, 5065, 5095	<code>\c_@@_tikz_loaded_bool</code>
<code>\g_@@_static_num_of_col_int</code>	27, 41, 1387, 2963, 5072, 5572
333, 1644, 1692, 5629, 6783, 6795, 6986	<code>\l_@@_tikz_seq</code>
<code>\l_@@_stop_loop_bool</code>	336, 5573, 5696, 5705
3156, 3157,	<code>\g_@@_total_X_weight_int</code>
3189, 3202, 3211, 3224, 3225, 3257, 3270, 3279	301, 1537, 1540, 1548, 1688, 2069
<code>\@@_store_in_tmpb_tl</code>	<code>\@@_true_c:</code>
5875, 5877	202, 218, 1753, 2177
<code>\@@_stroke_block:nnn</code>	<code>\l_@@_type_of_col_tl</code>
5665, 5879	826,
<code>\@@_stroke_borders_block:nnn</code> ...	827, 2849, 2851, 6075, 6076, 6077, 6085, 6090
5677, 5975	<code>\c_@@_types_of_matrix_seq</code>
<code>\@@_stroke_horizontal:n</code> ..	6738, 6739, 6744, 6748
6003, 6005, 6020	<code>\@@_update_for_first_and_last_row:</code> ..
<code>\@@_stroke_vertical:n</code>	930, 995, 1285, 2744, 2789
5999, 6001, 6009	<code>\@@_use_arraybox_with_notes:</code> ...
<code>\@@_sub_matrix:nnnnnnn</code>	1589, 2395
6365, 6369	<code>\@@_use_arraybox_with_notes_b:</code> ..
<code>\@@_sub_matrix_i:nnnn</code>	1586, 2379
6419, 6425	<code>\@@_use_arraybox_with_notes_c:</code>
<code>\l_@@_submatrix_extra_height_dim</code>	1587, 1618, 2323, 2393, 2432
349, 6284, 6453	<code>\@@_vdottedline:n</code>
<code>\l_@@_submatrix_hlines_clist</code>	2097, 5068
354, 6296, 6314, 6492, 6494	<code>\@@_vdottedline_i:n</code>
<code>\l_@@_submatrix_left_xshift_dim</code>	5075, 5080, 5084
350, 6286, 6505, 6538	<code>\@@_vline:nn</code>
<code>\l_@@_submatrix_name_str</code>	1829, 4572, 4708
6329, 6387, 6527, 6529, 6541, 6543, 6551, 6555	<code>\@@_vline_i:nn</code>
<code>\g_@@_submatrix_names_seq</code>	2957, 4577, 4581
324, 2979, 6326, 6330, 6930	<code>\@@_vline_i_complete:nn</code>
<code>\l_@@_submatrix_right_xshift_dim</code>	2957, 4691
351, 6288, 6514, 6548	<code>\@@_vline_ii:nnnn</code> ...
<code>\g_@@_submatrix_seq</code> ...	4604, 4615, 4648, 4692
332, 1293, 3306, 6352	<code>\l_@@_vlines_clist</code>
<code>\l_@@_submatrix_slim_bool</code> 6294, 6395, 6898	353,
<code>\l_@@_submatrix_vlines_clist</code>	586, 598, 603, 632, 1676, 1682, 1714, 1726,
355, 6298, 6316, 6475, 6477	2105, 2112, 2253, 2260, 2653, 2961, 4706, 4707
<code>\@@_succ:n</code>	<code>\l_@@_vpos_col_str</code>
161,	1845, 1848, 1850, 1855, 1877, 1890, 2064
165, 1107, 1113, 1118, 1119, 1139, 1829,	<code>\l_@@_vpos_of_block_tl</code>
1992, 2112, 2142, 2260, 2287, 2629, 2635,	344, 345,
2640, 2641, 2665, 2673, 2686, 2687, 2698,	5343, 5345, 5451, 5462, 5540, 5557, 5605, 5607
2702, 2707, 2708, 3396, 3473, 3563, 3622,	<code>\@@_w:</code>
4421, 4466, 4482, 4625, 4656, 4702, 4760,	1668, 1751, 2175
4791, 4835, 4860, 5007, 5009, 5011, 5013,	<code>\l_@@_weight_int</code> 2060, 2065, 2066, 2069, 2071
5054, 5093, 5255, 5259, 5269, 5273, 5729,	<code>\l_@@_width_dim</code>
5731, 5775, 5914, 5916, 6046, 6176, 6178, 6233	864, 1547, 2859, 2860, 2870, 2871
<code>\l_@@_suffix_tl</code>	<code>\g_@@_width_first_col_dim</code>
5217, 5228,	318, 1480, 1580, 2541, 2745, 2746
5238, 5241, 5290, 5298, 5299, 5317, 5325, 5326	<code>\g_@@_width_last_col_dim</code>
<code>\c_@@_table_collect_begin_tl</code> .	317, 1479, 1639, 2694, 2790, 2791
217, 242, 244	<code>\l_@@_width_used_bool</code>
<code>\c_@@_table_print_tl</code>	325, 865, 1535
219, 245, 246	<code>\@@_write_aux_for_cell_nodes:</code> ..
<code>\l_@@_tabular_width_dim</code>	2997, 3059
272, 1090, 1092, 1731, 2880	<code>\l_@@_x_final_dim</code>
<code>\l_@@_tabularnote_tl</code> 369, 843, 871, 2339, 2348	312, 3332,
<code>\g_@@_tabularnotes_seq</code>	3381, 3390, 3391, 3394, 3397, 3398, 3549,
368, 408, 2354, 2360, 2376	3565, 3573, 3577, 3581, 3583, 3588, 3590,
<code>\@@_test_hline_in_block:nnnn</code>	3624, 3633, 3641, 3681, 3689, 3728, 3743,
4725, 4727, 4863	3752, 3786, 3838, 3854, 4213, 5055, 5064,
<code>\@@_test_hline_in_stroken_block:nnnn</code> ..	5090, 6394, 6410, 6411, 6417, 6514, 6531, 6548
4729, 4885	<code>\l_@@_x_initial_dim</code>
<code>\@@_test_if_cell_in_a_block:nn</code>	310, 3327, 3359, 3368, 3369, 3372, 3375,
4941, 4959, 4977, 4997	3376, 3549, 3564, 3565, 3572, 3577, 3581,
<code>\@@_test_if_cell_in_block:nnnnnnn</code> ...	3583, 3585, 3588, 3590, 3615, 3633, 3641,
5003, 5005	3681, 3689, 3725, 3742, 3752, 3786, 3838,
<code>\@@_test_if_math_mode:</code>	3852, 3854, 3872, 3874, 4210, 5048, 5062,
277, 1487, 2820	5089, 6393, 6403, 6404, 6414, 6505, 6530, 6538
<code>\@@_test_in_corner_h:</code>	<code>\l_@@_xdots_color_tl</code> 519, 533, 3415, 3454,
4730, 4758	3535, 3536, 3604, 3656, 3708, 4080, 4155, 4172
	<code>\l_@@_xdots_down_tl</code> ...
	537, 3715, 3736, 3771

<code>\l_@@_xdots_line_style_tl</code>	494, 496, 529, 3700, 3708, 5065, 5095
<code>\l_@@_xdots_shorten_dim</code>	490, 491, 535, 2945, 3722, 3723, 3812, 3823, 3831
<code>\l_@@_xdots_up_tl</code>	538, 3714, 3735, 3761
<code>\l_@@_y_final_dim</code>	313, 3333, 3433, 3437, 3475, 3479, 3481, 3506, 3516, 3517, 3635, 3638, 3683, 3686, 3728, 3743, 3751, 3788, 3843, 3862, 4214, 5046, 5094, 6234, 6256, 6271, 6431, 6446, 6447, 6452, 6470, 6487, 6531, 6539, 6549
<code>\l_@@_y_initial_dim</code>	311, 3328, 3426, 3436, 3474, 3475, 3479, 3481, 3488, 3498, 3499, 3635, 3640, 3683, 3688, 3725, 3742, 3751, 3788, 3843, 3860, 3862, 3872, 3875, 4211, 5044, 5045, 5046, 5092, 6232, 6256, 6271, 6428, 6439, 6440, 6452, 6469, 6486, 6530, 6539, 6549
<code>\l</code>	2487, 2509, 6097, 6103, 6109, 6657, 6658, 6698, 6707, 6735, 6795, 6800, 6805, 6810, 6815, 6859, 6864, 6870, 6877, 6883, 6884, 6899, 6904, 6912, 6918, 6924, 6925, 6936, 6948, 6955, 6967, 6974, 6981, 6990, 6996, 7003, 7010, 7016, 7022, 7034, 7038, 7043, 7049, 7058, 7059, 7073, 7074, 7090, 7094, 7095, 7113, 7114, 7156, 7157, 7208, 7209, 7259, 7260, 7313, 7314
<code>\{</code>	293, 1756, 1993, 2018, 2827, 6126, 6510, 6917, 7003, 7156, 7259
<code>\}</code>	293, 1759, 1993, 2003, 2827, 6126, 6519, 6917, 7003, 7156, 7259
<code>\ </code>	2829, 6125
<code>\sqcup</code> ..	6734, 6762, 6767, 6774, 6782, 6783, 6794, 6795, 6858, 6876, 6877, 6890, 6896, 6900, 6908, 6911, 6923, 6929, 6941, 6985, 6986, 6987, 6995, 7001, 7008, 7021, 7028, 7029, 7037
A	
<code>\A</code>	6324
<code>\aboverulesep</code>	2370
<code>\addtocounter</code>	425
<code>\alph</code>	371
<code>\anchor</code>	3056, 3057
<code>\arraybackslash</code>	1910, 2080, 2210
<code>\arraycolor</code>	1401, 6734
<code>\arraycolsep</code>	616, 618, 620, 1089, 1183, 1305, 1306, 1617, 1621, 2328, 2664, 2668, 2678, 5051, 5058
<code>\arrayrulecolor</code>	107
<code>\arrayrulewidth</code>	140, 145, 157, 546, 921, 1106, 1108, 1114, 1145, 1679, 1685, 1772, 1822, 2108, 2115, 2256, 2263, 2387, 2426, 2553, 2555, 2561, 2571, 2573, 2579, 2602, 2604, 2610, 2628, 2630, 2636, 2662, 2666, 2678, 2681, 4464, 4465, 4467, 4483, 4485, 4667, 4668, 4670, 4681, 4687, 4803, 4814, 4820, 4856, 5130, 5883, 5938, 5963, 5965, 5977, 6256, 6453, 6457
<code>\arraystretch</code>	1182, 3489, 3507, 5427, 5533, 5549, 6429, 6432
<code>\AtBeginDocument</code>	23, 28, 85, 101, 177, 183, 227, 375, 489, 491, 493, 505, 695, 711, 3024, 3879, 4009, 4086, 4164, 4197, 5029, 6358
<code>\AtBeginEnvironment</code>	131, 1248
<code>\AutoNiceMatrix</code>	6127
<code>\AutoNiceMatrixWithDelims</code> ...	6079, 6119, 6131
B	
<code>\baselineskip</code>	110, 116, 1935
<code>\begingroup</code>	2129
<code>\bgroup</code>	1474
<code>\bigskip</code>	1524, 3104
<code>\Block</code>	1237, 6954, 6961, 7008, 7049
<code>\BNiceMatrix</code>	6065
<code>\bNiceMatrix</code>	6062
<code>\Body</code>	1322
bool commands:	
<code>\bool_do_until:Nn</code>	3157, 3225
<code>\bool_gset_false:N</code>	959, 1006, 1007, 1128, 1257, 1409, 1481, 1506, 1675, 2756, 2803, 4872, 4883, 4894, 4905, 5476
<code>\bool_gset_true:N</code>	1509, 1832, 2546, 2720, 2765, 2856, 3897, 3913, 3929, 3953, 3976, 3987, 4161, 4587, 4723
<code>\bool_if:NTF</code>	168, 697, 702, 713, 718, 888, 891, 987, 1102, 1129, 1176, 1180, 1185, 1245, 1246, 1275, 1280, 1294, 1385, 1387, 1411, 1414, 1416, 1436, 1473, 1488, 1498, 1535, 1559, 1637, 1642, 1661, 1666, 1694, 1706, 1957, 1973, 2002, 2236, 2366, 2532, 2549, 2567, 2585, 2598, 2624, 2658, 2690, 2695, 2724, 2742, 2770, 2787, 2889, 2891, 2893, 2926, 2941, 2963, 2980, 2983, 2997, 3478, 3480, 3627, 3675, 3895, 3911, 3927, 3951, 3974, 4349, 4372, 4950, 4968, 4986, 5113, 5123, 5145, 5430, 5471, 5536, 5552, 5651, 5721, 5738, 5782, 5808, 5845, 6138, 6710, 6720, 6753, 6898
<code>\bool_if:nTF</code>	185, 377, 404, 1065, 1569, 3311, 4186, 4427, 4697, 4701, 4830, 4834, 5139, 5388, 5835, 6235, 6245, 6247, 6260, 6266, 6275
<code>\bool_lazy_all:nTF</code>	1710, 1722, 2649, 2950, 4658, 4793, 4865, 4876, 4887, 4898
<code>\bool_lazy_and:nnTF</code>	230, 1776, 2325, 2588, 2663, 2667, 2675, 2725, 3467, 3734, 3993, 4438, 5433, 5901, 6479, 6496
<code>\bool_lazy_or:nnTF</code>	526, 999, 1057, 1702, 2308, 2337, 2415, 2773, 3544, 3792, 4419, 4451, 4455, 4505, 4509, 4942, 4960, 4978, 5363, 5368, 6382
<code>\bool_lazy_or_p:nn</code>	2728
<code>\bool_not_p:n</code>	1713, 1715, 1725, 1727, 2590, 2652, 2654
<code>\bool_set:Nn</code>	3548
<code>\c_false_bool</code>	2013, 2021, 2034, 2040, 2046, 3891, 3907, 3923
<code>\g_tmpa_bool</code>	4587, 4596, 4630, 4638, 4643, 4723, 4731, 4765, 4773, 4778, 4872, 4883, 4894, 4905
<code>\g_tmpb_bool</code>	1675, 1706, 1832
<code>\l_tmpb_bool</code> ...	4947, 4961, 4979, 5001, 5014
<code>\c_true_bool</code>	1992

box commands:		
\box_clear_new:N	1178, 1299	
\box_dp:N	915, 935, 972, 992, 1047, 1206, 1215, 1290, 2215, 2277, 2445, 2458, 3507, 5506, 6432	
\box_gclear_new:N	5417	
\box_grotate:Nn	5473	
\box_ht:N	916, 937, 943, 955, 978, 990, 1039, 1208, 1210, 1213, 1288, 1905, 1926, 1928, 1934, 2211, 2276, 2445, 2458, 3489, 5497, 6429	
\box_move_down:nn	1047, 1932	
\box_move_up:nn	76, 78, 80, 1039, 2320, 2393, 2432	
\box_rotate:Nn	949	
\box_set_dp:Nn	971, 991, 2277	
\box_set_ht:Nn	977, 989, 2276	
\box_set_wd:Nn	965, 2327	
\box_use:N	428, 956, 1046, 1937	
\box_use_drop:N	997, 1003, 1022, 1959, 2238, 2279, 2320, 2321, 2333, 2751, 5516, 5830, 5867	
\box_wd:N	429, 966, 994, 1001, 1060, 1310, 1312, 1547, 2328, 2330, 2452, 2464, 2746, 2749, 2791, 2795, 5485, 6145	
\l_tmpa_box	411, 428, 429, 1309, 1310, 1311, 1312, 1604, 2276, 2277, 2279, 2320, 2321, 2445, 2458	
\l_tmpb_box	2438, 2452, 2453, 2464	
C		
\c	59, 1698	
\Cdots	1228, 3902, 3905	
\cdots	1161, 1220	
\cellcolor	1170, 1395, 4525	
\centering	1885, 5441	
char commands:		
\char_set_catcode_space:n	1654	
\chessboardcolors	1403	
\cline	160, 1225, 1226	
clist commands:		
\clist_clear:N	4498	
\clist_if_empty:NnTF	4594, 4730, 5673	
\clist_if_empty_p:N	2653	
\clist_if_in:NnTF	1137, 1682, 2112, 2260, 4707, 4840, 5998, 6000, 6002, 6004, 6023	
\clist_if_in:nnTF	5984	
\clist_map_inline:Nn	4445, 4468, 4499, 4910, 5982, 6477, 6494	
\clist_map_inline:nn	2844, 4305, 4357, 6043	
\clist_new:N	337, 352, 353, 354, 355, 503	
\clist_put_right:Nn	4516	
\clist_set:Nn	598, 599, 603, 604, 631, 632, 633	
\clist_set_eq:NN	4497	
\l_tmpa_clist	4497, 4499	
\CodeAfter	883, 1241, 2490, 2493, 2719, 2764, 2978, 7061	
\CodeBefore	1470, 6734	
\color	111, 117, 172, 174, 1608, 1626, 3409, 3412, 3415, 3448, 3451, 3454, 3529, 3532, 3536, 3604, 3656, 4074, 4077, 4080, 4149, 4152, 4155, 4172, 4237, 5424, 5582, 6265, 6612, 6636	
\colorlet	285, 286, 899, 906, 2734, 2778	
\columncolor	1172, 1402, 2989, 4548	
\cr	144, 162, 2713	
\crrr	2526	
cs commands:		
\cs_generate_variant:Nn	58, 164, 3731, 4229, 4230, 5135, 5136	
\cs_gset:Npn	111, 117, 5128	
\cs_gset_eq:NN	69, 247, 1199, 1490, 3000	
\cs_if_exist:NnTF	57, 131, 188, 1268, 1270, 1451, 1491, 1494, 3046, 3047, 3068, 3192, 3205, 3260, 3273, 3362, 3384, 3492, 3510, 4042, 4060, 4117, 4135, 5115, 5168, 5743, 5761, 6240, 6399, 6406, 6435, 6442	
\cs_if_exist_p:N	232, 527, 4661, 4796, 4944, 4963, 4981	
\cs_if_free:NnTF	65, 3404, 3443, 3524, 3599, 3651	
\cs_if_free_p:N	4188, 4190	
\cs_new_protected:Npx	3026, 4199, 5031	
\cs_set:Nn	683, 686, 689	
\cs_set:Npn	107, 108, 113, 114, 119, 132, 133, 147, 149, 150, 172, 174, 374, 1153, 2130, 2152, 3151, 3213, 3281, 4084, 4159, 4844, 4845, 4851, 4852, 5427, 5533, 5549	
\cs_set_nopar:Npn	1079, 1091, 1182, 1193, 5310, 5311	
\cs_set_nopar:Npx	1092	
\cs_set_protected:Npn	6116	
\cs_set_protected_nopar:Npn	5405, 5708	
D		
\Ddots	1230, 3935, 3936, 3941, 3942	
\ddots	1163, 1222	
\diagbox	1242, 5405, 5708	
dim commands:		
\dim_add:Nn	5279	
\dim_compare:nNnTF	110, 116, 963, 969, 975, 1731, 2586, 2795, 3372, 3394, 3583, 5192, 5202, 5753, 5773, 6145	
\dim_compare_p:n	5435	
\dim_gzero:N	967, 973, 979	
\dim_max:nn	5181, 5185	
\dim_min:nn	5173, 5177	
\dim_ratio:nn	3642, 3690, 3806, 3811, 3822, 3830, 3839, 3844, 3855, 3863	
\dim_set:Nn	5154, 5161, 5172, 5176, 5180, 5184, 5196, 5197, 5206, 5207, 5251, 5264	
\dim_set_eq:NN	5152, 5159, 5259, 5273	
\dim_sub:Nn	5276	
\dim_use:N	5193, 5203, 5254, 5255, 5267, 5268, 5291, 5292, 5293, 5294, 5318, 5319, 5320, 5321	
\dim_zero_new:N	5151, 5153, 5158, 5160	
\c_max_dim	3359, 3372, 3381, 3394, 5152, 5154, 5159, 5161, 5193, 5203, 5740, 5753, 5758, 5773, 6236, 6237, 6393, 6394, 6414, 6417	
\dotfill	1240, 6134	
\dots	1165	
\doublerulesep	1823, 4670, 4682, 4803, 4815, 4857	
\doublerulesepcolor	113	
\draw	3719	
E		
\egroup	1652	
\else	2130	

else commands:		
\else:	279	
\endarray	2478, 2506	
\endBNiceMatrix	6066	
\endbNiceMatrix	6063	
\endgroup	2138	
\endNiceArray	2866, 2876, 2885	
\endNiceArrayWithDelims	2813, 2823	
\endpgfscope	3082, 3095, 3776, 6191	
\endpNiceMatrix	6054	
\endsavenotes	1661	
\endtabularnotes	2361	
\endVNiceMatrix	6060	
\endvNiceMatrix	6057	
\enskip	1983, 2011, 2019, 2032, 2038	
\ensuremath	3896, 3912, 3928, 3952, 3975	
\everycr	143, 162, 1203	
\everypar	1903, 1906	
exp commands:		
\exp_after:wN	194, 214, 1523, 1609, 1627, 1691, 2135, 6459	
\exp_args:NNc	5117	
\exp_args:Nne	2851	
\exp_args:NNV	2498, 3883, 3899, 3915, 3931, 3955, 4013, 4090, 4168, 6362	
\exp_args:NNx	1136, 2111, 2259	
\exp_args:No	3707	
\exp_args:NV	1670, 2136, 2474, 2501, 2503, 5892	
\exp_args:Nxx	5398, 5399	
\exp_last_unbraced:NV	1412, 2981, 5685	
\exp_not:N	42, 43, 46, 47, 1772, 1816, 1881, 1882, 1885, 1886, 1887, 2897, 2911, 2919, 2921, 2992, 4536, 4548, 4929, 5033, 5034, 5451, 5462, 5540, 5557, 5688, 6089, 6565	
\exp_not:n	1071, 1658, 2993, 4023, 4024, 4100, 4525, 4536, 4538, 4549, 5414, 5514, 5526, 5527, 5656, 5666, 5678, 5690, 5717, 6091, 6155, 6156	
\ExplSyntaxOff	67, 1660, 5132	
\ExplSyntaxOn	64, 1653, 5125	
\extrarowheight	3489, 5428, 5534, 5550, 6429	
F		
\fi	121, 1672, 2130, 2133, 4844, 4861	
fi commands:		
\fi:	281	
\five	3051, 3056	
flag commands:		
\flag_clear_new:n	6558	
\flag_height:n	6583	
\flag_raise:n	6582	
\fontdimen	2316	
fp commands:		
\fp_eval:n	3747	
\fp_to_dim:n	3782	
\futurelet	126	
G		
\globaldefs	1648, 5634	
group commands:		
\group_insert_after:N	6139, 6140, 6142, 6143	
H		
\halign	1217	
\hbox	1100, 1613, 2393, 2432, 2551, 2569, 2596, 2600, 2626, 2660, 3077, 3090	
hbox commands:		
\hbox:n	76, 78, 81, 2331	
\hbox_gset:Nn	5419	
\hbox_overlap_left:n	1040, 1048, 2530, 2747	
\hbox_overlap_right:n	428, 2692, 2793	
\hbox_set:Nn	411, 1037, 1309, 1311, 1604, 1930, 2079, 2438, 2453, 5720	
\hbox_set:Nw	887, 1314, 1948, 2227, 2722, 2768	
\hbox_set_end:	985, 1534, 1956, 2235, 2741, 2786	
\hbox_to_wd:nn	453, 478	
\Hdotsfor	1235, 6762, 6941	
\hdotsfor	1166	
\hdottedline	1232	
\heavyrulewidth	2371	
\hfil	1752, 2176	
\hfill	140, 157	
\Hline	1233, 4847	
\hline	119	
\hrule	123, 140, 157, 1145, 2371, 6270, 6617, 6641	
\hskip	122	
\Hspace	1234	
\hspace	3988	
\hss	1752, 2176	
I		
\ialign	1097, 1193, 1216, 5613	
\iddots	1231, 3959, 3960, 3965, 3966	
\iddots	71, 1164, 1223	
if commands:		
\if_mode_math:	279	
\IfBooleanTF	1525	
\ifnum	121, 4844, 4861	
\ifstandalone	1494	
\ignorespaces	1360, 2159	
int commands:		
\int_case:nnTF	3933, 3939, 3957, 3963	
\int_compare:nNnTF	135, 136, 152, 885, 886, 896, 903, 940, 950, 1142, 1144, 1272, 1278, 1283, 1537, 1540, 1557, 1561, 1572, 1576, 1577, 1644, 1925, 2139, 2157, 2349, 2388, 2427, 2499, 2527, 2771, 3167, 3174, 3178, 3180, 3235, 3242, 3246, 3248, 3411, 3450, 3531, 3566, 3568, 4076, 4151, 4414, 4459, 4477, 4513, 4544, 4561, 4562, 4569, 4570, 4574, 5007, 5009, 5011, 5013, 5423, 5425, 5478, 5490, 5801, 5833, 5837, 5910, 5912, 6093, 6095, 6097, 6101, 6103, 6105, 6107, 6109, 6463, 6465	
\int_compare_p:n	3313, 3314, 3315, 3316, 4429, 4431, 5902, 5903	
\int_do_until:nNnn	4369	
\int_gadd:Nn	2069, 2156	
\int_gincr:N	884, 913, 1497, 1796, 1892, 1962, 1968, 2088, 2622, 2647, 2766, 3629, 3677, 5110, 5404	
\int_if_even:nTF	4314, 6583	
\int_incr:N	407, 1806, 4397	
\int_min:nn	3114, 3116, 3118, 3120, 3130, 3132, 3321, 3322	
\int_mod:nn	4385	

`\pgfsetbaseline` 1012
`\pgfsetcornersarced` 4486, 5894
`\pgfsetlinewidth`
 4687, 4820, 5921, 5950, 5997, 6457
`\pgfsetrectcap` 4688, 4821
`\pgfsetroundcap` 6183
`\pgfsetstrokecolor` 5892
`\pgfsyspdfmark` 65, 66
`\pgftransformrotate` 3745
`\pgftransformshift` 444, 469, 3071, 3084, 3122,
 3739, 5818, 5840, 6188, 6192, 6252, 6535, 6545
`\pgfusepath` 3765, 3775, 4240, 4674, 5922
`\pgfusepathqfill` 3877, 4807
`\pgfusepathqstroke` 4689, 4822,
 5960, 5971, 6018, 6031, 6184, 6471, 6488, 6522
`\phantom` 3896, 3912, 3928, 3952, 3975
`\pNiceMatrix` 6053
prg commands:
 `\prg_do_nothing:`
 69, 186, 229, 238, 247, 539, 1199, 2978
 `\prg_new_conditional:Nnn` 4417, 4425
 `\prg_replicate:nn` 417, 2617,
 2618, 4027, 4679, 4812, 6096, 6099, 6102, 6108
 `\prg_return_false:` 4422, 4434
 `\prg_return_true:` 4423, 4433
`\ProcessKeysOptions` 6691
`\ProvideDocumentCommand` 71
`\ProvidesExplPackage` 4

Q

`\quad` 398
quark commands:
 `\q_stop` 132, 133,
 149, 150, 171, 173, 1412, 1434, 1523, 1691,
 1761, 1831, 2006, 2028, 2135, 2178, 2490,
 2493, 4162, 4176, 4177, 4244, 4293, 4298,
 4361, 4449, 4450, 4472, 4473, 4503, 4504,
 5308, 5315, 5355, 5356, 5360, 5685, 5877,
 5900, 5909, 5940, 5943, 5987, 5990, 6068,
 6083, 6346, 6351, 6376, 6379, 6459, 6568, 6576

R

`\raggedleft` 1887, 5442
`\raggedright` 1886, 5443
`\rectanglecolor` 1396, 2988, 4536
`\refstepcounter` 426
regex commands:
 `\regex_const:Nn` 59
 `\regex_match:nnTF` 6324
 `\regex_replace_all:NnN` 1696
`\relax` 165, 166
`\renewcommand` 192, 212
`\RenewDocumentEnvironment`
 6052, 6055, 6058, 6061, 6064
`\RequirePackage` 1, 3, 9, 10, 11, 131
`\right` 1627, 2448, 2460, 6275, 6621, 6645
`\rotate` 1238
`\roundedrectanglecolor` 1397, 5688
`\rowcolor` 1171, 1398
`\rowcolors` 1399
`\rowlistcolors` 1400
`\RowStyle` 1244

S

`\savedanchor` 3051
`\savenotes` 1473
scan commands:
 `\scan_stop:` 2982
`\scriptstyle`
 891, 2724, 2770, 3726, 3727, 3761, 3771
seq commands:
 `\seq_clear:N` 1689
 `\seq_clear_new:N` 4342, 4909
 `\seq_count:N` 2500, 4225, 4387
 `\seq_gclear:N` 1250, 1251,
 1293, 1295, 1500, 1501, 1502, 1503, 2376, 2979
 `\seq_gclear_new:N` 1296, 1297, 1408, 2496, 2512
 `\seq_gpop_left:NN` 2502, 2514
 `\seq_gput_left:Nn` 662, 2141, 2143, 5648
 `\seq_gput_right:Nn` 408, 1769, 2144,
 3292, 4224, 5511, 5523, 5670, 6158, 6330, 6352
 `\seq_gset_from_clist:Nn`
 2897, 2911, 2919, 2921
 `\seq_gset_map_x:NNn` 3006
 `\seq_gset_split:Nnn` 2498, 2513
 `\seq_if_empty:NTF` 2335, 2907, 2915, 4925, 5696
 `\seq_if_empty_p:N` ... 2952, 2953, 2954, 4439
 `\seq_if_in:NnTF`
 660, 4479, 4627, 4633, 4640, 4762,
 4768, 4775, 5174, 5182, 5746, 5764, 6326, 6748
 `\seq_item:Nn`
 1276, 1281, 1364, 1365, 1366, 1367, 4406, 4408
 `\seq_map_function:NN` 2504
 `\seq_map_indexed_inline:Nn` 4220, 4235
 `\seq_map_inline:Nn`
 2354, 2360, 2516, 3306, 4376, 4588,
 4590, 4592, 4724, 4726, 4728, 5002, 5614, 6461
 `\seq_mapthread_function:NNN` 5303
 `\seq_new:N` 268, 284, 319, 320, 321,
 322, 323, 324, 326, 327, 332, 336, 368, 6738
 `\seq_put_right:Nn` 4989, 5573
 `\seq_set_eq:NN` 4351
 `\seq_set_filter:NNn` 4352, 4374
 `\seq_set_from_clist:Nn` 4929, 6739
 `\seq_set_map_x:NNn` 6744
 `\seq_set_split:Nnn` 4343
 `\seq_use:Nn` 5705
 `\seq_use:Nnnn`
 2912, 2920, 2922, 4930, 6930, 7318
 `\l_tmpa_seq` 4352, 4374
 `\l_tmpb_seq` 4351, 4352, 4374, 4376
`\setlist` 384, 395, 698, 703, 714, 719
siunitx commands:
 `\siunitx_cell_begin:w` 188, 200, 232
 `\siunitx_cell_end:` 203
skip commands:
 `\skip_gadd:Nn` 2593
 `\skip_gset:Nn` 2584
 `\skip_gset_eq:NN` 2591, 2592
 `\skip_horizontal:N` 141, 158, 1315,
 1316, 1532, 1533, 1579, 1580, 1616, 1617,
 1620, 1621, 1639, 1640, 1679, 1685, 1772,
 2095, 2108, 2115, 2256, 2263, 2465, 2466,
 2468, 2469, 2540, 2541, 2553, 2555, 2571,
 2573, 2595, 2602, 2604, 2623, 2628, 2630,

2648, 2657, 2662, 2664, 2666, 2668, 2694, 2752, 2753, 2754, 2757, 2792, 2797, 2798, 2799	
\skip_horizontal:n	429, 1060, 1818
\skip_vertical:N	
145, 1106, 1108, 1612, 1623, 2345, 2370, 5021	
\skip_vertical:n	955, 4854
\g_tmpa_skip	
... 2584, 2591, 2592, 2593, 2595, 2623, 2648	
\c_zero_skip	1204
\smash	6646, 6647
\space	292, 293
\stepcounter	415, 420
str commands:	
\c_backslash_str	292
\c_colon_str	815
\str_case:nn	
1883, 5439, 5811, 5823, 5848, 5860, 6506, 6515	
\str_case:nnTF	
... 1584, 1739, 2163, 2302, 4912, 6580, 6593	
\str_clear_new:N	6387
\str_gclear:N	2998
\str_gset:Nn	
1483, 2810, 2819, 2848, 2861, 2869, 2879, 6118	
\str_if_empty:NTF	
... 922, 1025, 1115, 1482, 2537, 2562, 2580, 2611, 2637, 2683, 2704, 2809, 2818, 3126, 3138, 5295, 5322, 5375, 6527, 6541, 6551	
\str_if_eq:nnTF	96,
291, 1095, 1764, 1767, 1811, 1831, 1860, 1877, 1880, 1890, 1976, 2008, 2030, 2053, 2102, 2250, 2485, 2487, 2520, 4406, 5890, 6220	
\str_if_eq_p:nn	528, 1703,
1704, 1777, 4453, 4457, 4507, 4511, 5365, 5370	
\str_if_in:NnTF	2290, 2402
\str_new:N	263, 287, 341, 509, 814
\str_range:Nnn	2294, 2406
\str_set:Nn	
... 264, 342, 659, 803, 1792, 1839, 1841, 1843, 1845, 1848, 1850, 1855, 1868, 1881, 1882, 1950, 2063, 2186, 2229, 5331, 5333, 5335, 5337, 5339, 5341, 5343, 5345, 5376, 5379, 5430, 5537, 5553, 5590, 5592, 5594, 5596, 5599, 5602, 5605, 5607, 5834, 5838, 6329	
\str_set_eq:NN	663, 815, 5377
\l_tmpa_str	659, 660, 662, 663
\strut	2354, 2360
\strutbox	3489, 3507, 6429, 6432
\SubMatrix	1404, 2962, 6355, 6858, 6883, 6890, 6896, 6900, 6923, 7076
sys commands:	
\sys_if_engine_xetex_p:	1057
\sys_if_output_dvi_p:	1057
T	
\tabcolsep	1088, 1616, 1620
\tabskip	1204
\tabularnote	379, 402, 409, 7001, 7021
\tabularnotes	2359
TeX and L ^A T _E X 2 _ε commands:	
\@BTnormal	1177
\@acol	1078
\@acoll	1076
\@acolr	1077
\@addamp	2130
\@addtopreamble	2137
\@array@array	1080
\@arrayacol	1076, 1077, 1078
\@arstrut	2153
\@arstrutbox	
... 915, 916, 955, 1206, 1208, 1210, 1213, 1215, 1905, 1914, 1928, 1934, 2211, 2215	
\@currenvir	6220
\@depth	6272, 6618, 6642
\@empty	2137
\@finalstrut	1914
\@firstampfalse	2130
\@gobblethree	66
\@halignto	1079, 1091, 1092
\@height	124, 140, 157, 6270, 6617, 6641
\@ifclassloaded	51, 54, 6712, 6722
\@ifnextchar	1255
\@ifpackageloaded	
... 30, 33, 36, 39, 87, 103, 179, 6715, 6725	
\@mainaux	62,
88, 1653, 1654, 1655, 1660, 5125, 5126, 5132	
\@mkpream	2136
\@preamble	2154
\@preamerr	2130
\@sharp	2152
\@tabarray	1093
\@tempswafalse	1672, 2133
\@tempswatrue	1671, 2132
\@temptokena	194, 196, 214, 216, 237, 240, 1670, 1691, 2131, 2135
\@whiles w	1672, 2133
\@width	124, 6273, 6619, 6643
\@xhline	127
\bBigg@	1309, 1311
\c@MaxMatrixCols	2838, 6776
\c@tabularnote	2338, 2349, 2377
\col@sep	1088, 1089, 1579, 1640, 2540, 2593, 2657, 2757, 2792, 3376, 3398
\CT@arc	107, 108
\CT@arc@	106, 111, 125, 139, 156, 172, 174, 1490, 2371, 3000, 4686, 4819, 5086, 5891, 5949, 5996, 6182, 6460
\CT@drsc	113, 114
\CT@drsc@	
... 117, 4661, 4662, 4666, 4796, 4797, 4801	
\CT@everycr	1197
\CT@row@color	1199
\if@firstamp	2130
\if@tempswa	1672, 2133
\NC@	1153
\NC@find	205, 221, 238
\NC@list	1672, 2133
\NC@rewrite@S	192, 212, 239
\new@ifnextchar	1255
\newcol@	1155, 1156
\nicematrix@redefine@check@rerun	88, 91
\pgf@relevantforpicturesizefalse	
... 1371, 3036, 3063, 3699, 3868, 4234, 4356, 4651, 4786, 5215, 5225, 5236, 5724, 5887, 5948, 5995, 6169, 6230, 6391	
\pgfsys@getposition	
... 1369, 1374, 1380, 1446, 1454, 1457	

<code>\pgfsys@markposition</code>		<code>\tl_new:N</code>	242,
.... 1042, 1050, 1107, 1189, 1368, 2533,		245, 276, 283, 295, 296, 297, 300, 307, 309,	
2554, 2572, 2603, 2629, 2665, 2697, 3078, 3091		334, 335, 339, 344, 369, 494, 498, 517, 519, 520	
<code>\pgfutil@check@rerun</code>	93, 94	<code>\tl_put_left:Nn</code>	1177, 1417
<code>\reserved@a</code>	126	<code>\tl_put_right:Nn</code>	642, 1187, 1285, 1324, 1516
<code>\rvtx@ifformat@geq</code>	57	<code>\tl_range:nnn</code>	96
<code>\set@color</code>	5423, 5720	<code>\tl_set:Nn</code> ..	243, 4299, 4300, 4362, 4378,
<code>\tikz@library@external@loaded</code>	1491	4381, 4458, 4460, 4476, 4478, 4512, 4514,	
tex commands:		4583, 5381, 5911, 5913, 5944, 5945, 5991, 5992	
<code>\tex_mkern:D</code>	75, 77, 79, 82	<code>\tl_set_eq:NN</code>	496, 4296, 4297,
<code>\tex_the:D</code>	196, 216	4461, 4599, 4734, 5065, 5095, 5941, 5942,	
<code>\textfont</code>	2316	5988, 5989, 6349, 6350, 6377, 6378, 6380, 6381	
<code>\textit</code>	371	<code>\tl_set_rescan:Nnn</code>	
<code>\textsuperscript</code>	372, 373 2497, 3882, 4012, 4089, 4167, 6361	
<code>\the</code>	165, 166, 1672, 1691, 2133, 2135	<code>\tl_to_str:n</code>	6745
<code>\thetabularnote</code>	374	<code>\g_tmpa_tl</code>	240, 243, 246
<code>\tikzexternaldisable</code>	1493	tmpc commands:	
<code>\tikzset</code>	1389, 1495, 2965	<code>\l_tmpc_dim</code>	314, 3119, 3123,
tl commands:		4464, 4465, 4488, 4657, 4668, 4673, 4678,	
<code>\tl_clear:N</code>	4609, 4620, 4744, 4755, 5882	4684, 4792, 4806, 4811, 4817, 5730, 5736,	
<code>\tl_clear_new:N</code> .	4294, 4295, 4344, 4380,	5780, 5908, 5919, 6014, 6017, 6177, 6180, 6188	
4584, 4720, 6347, 6348, 6372, 6373, 6374, 6375		<code>\l_tmpc_int</code>	4367, 4368, 4369
<code>\tl_const:Nn</code>		<code>\l_tmpc_tl</code>	4294, 4296, 4299,
..... 42, 43, 46, 47, 495, 2715, 2760, 6569		4461, 4480, 4584, 4598, 4599, 4602, 4607,	
<code>\tl_count:n</code>	2297, 2409	4609, 4613, 4618, 4620, 4720, 4733, 4734,	
<code>\tl_gclear:N</code>	882,	4737, 4742, 4744, 4748, 4753, 4755, 5941,	
912, 1505, 1512, 1674, 2134, 2977, 2984, 4239		5953, 5966, 5988, 6005, 6011, 6347, 6349, 6353	
<code>\tl_gclear_new:N</code>		tmpd commands:	
1258, 1259, 1260, 1261, 1262, 1263, 1264, 1504		<code>\l_tmpd_dim</code>	
<code>\tl_gput_left:Nn</code>	1065, 1708, 4546 315, 3121, 3124, 4485, 4488, 4669,	
<code>\tl_gput_right:Nn</code>	1065, 1551, 1720, 1771,	4673, 4802, 4806, 5732, 5736, 5758, 5769,	
1814, 1828, 1991, 2012, 2020, 2033, 2039,		5773, 5776, 5780, 5917, 5920, 6179, 6180, 6192	
2045, 2096, 2895, 2909, 2917, 2990, 4015,		<code>\l_tmpd_tl</code>	4295, 4297, 4300, 5942,
4092, 4227, 4523, 4534, 4859, 4927, 5026,		5955, 5964, 5989, 6001, 6022, 6348, 6350, 6353	
5407, 5653, 5663, 5675, 5686, 5698, 5710, 6148		token commands:	
<code>\tl_gset:Nn</code>	240,	<code>\token_to_str:N</code>	6733, 6734, 6762,
244, 246, 1357, 1475, 1476, 1477, 1657,		6853, 6858, 6864, 6882, 6890, 6896, 6900,	
1678, 1684, 1978, 1979, 2009, 2035, 2992, 4225		6917, 6923, 6941, 6954, 6960, 7001, 7008,	
<code>\tl_if_blank:nTF</code>	4257, 4260,	7021, 7038, 7048, 7060, 7069, 7075, 7113, 7313	
4266, 4269, 4275, 4278, 4284, 4287, 4394, 5354			
<code>\tl_if_blank_p:n</code>			
4452, 4456, 4506, 4510, 4662, 4797, 5364, 5369			
<code>\tl_if_empty:NnTF</code>	1132, 1513, 1522, 1607,		
1625, 2348, 2960, 2961, 2986, 4391, 4474,			
4475, 4598, 4602, 4613, 4733, 4737, 4748,			
5422, 5661, 5683, 5888, 6264, 6458, 6611, 6635			
<code>\tl_if_empty:nTF</code>	640, 778, 820, 836, 852,		
873, 2482, 2509, 3415, 3454, 3535, 3604,			
3656, 4080, 4155, 4172, 6321, 6578, 6646, 6761			
<code>\tl_if_empty_p:N</code>	1714, 1726, 3735, 3736		
<code>\tl_if_empty_p:n</code>	2339, 5395		
<code>\tl_if_eq:NnTF</code>	3700		
<code>\tl_if_eq:NnTF</code> ..	1134, 1676, 2105, 2253,		
2278, 2397, 4706, 4839, 5061, 5063, 6475, 6492			
<code>\tl_if_eq:nnTF</code> ...	652, 794, 2006, 2028, 4221		
<code>\tl_if_exist:NnTF</code>	1507		
<code>\tl_if_in:NnTF</code>	4360, 4448, 4471, 4502		
<code>\tl_if_in:nnTF</code>	1993, 2003, 2018		
<code>\tl_if_single_token:nTF</code>	591, 802		
<code>\tl_if_single_token_p:n</code>	58		
<code>\tl_item:Nn</code>	243, 244, 246		
<code>\tl_map_inline:nn</code>	2483		

U

<code>\unskip</code>	2364
use commands:	
<code>\use:N</code>	1068,
1319, 1320, 1510, 1529, 1530, 2833, 2853, 4238	
<code>\use:n</code>	1874,
4173, 4553, 5449, 5460, 5538, 5555, 6087, 6564	
<code>\usepackage</code>	6717, 6727
<code>\usepgfmodule</code>	2

V

vbox commands:	
<code>\vbox:n</code>	81
<code>\vbox_set_top:Nn</code>	952
<code>\vbox_to_ht:nn</code>	449, 476, 2444, 2457
<code>\vbox_to_zero:n</code>	954
<code>\vcenter</code>	1610, 2442, 6267, 6614, 6638, 7038
<code>\Vdots</code>	1229, 3918, 3921
<code>\vdots</code>	1162, 1221
<code>\Vdotsfor</code>	1236
<code>\vfill</code>	452, 478
<code>\vline</code>	125
<code>\VNiceMatrix</code>	6059
<code>\vNiceMatrix</code>	6056

<code>\vrule</code>	123, 1905, 2211, 2215	X
<code>\vskip</code>	122	<code>\xglobal</code> 899, 906, 2734, 2778
<code>\vtop</code>	1104	Z
		<code>\Z</code> 6324

Contents

1	The environments of this package	2
2	The vertical space between the rows	2
3	The vertical position of the arrays	3
4	The blocks	4
4.1	General case	4
4.2	The mono-column blocks	5
4.3	The mono-row blocks	6
4.4	The mono-cell blocks	6
4.5	Horizontal position of the content of the block	7
5	The rules	7
5.1	Some differences with the classical environments	7
5.1.1	The vertical rules	7
5.1.2	The command <code>\cline</code>	8
5.2	The thickness and the color of the rules	8
5.3	The tools of nicematrix for the rules	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>	9
5.3.3	The (empty) corners	10
5.4	The command <code>\diagbox</code>	11
5.5	Dotted rules	11
6	The color of the rows and columns	11
6.1	Use of <code>colortbl</code>	11
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	12
6.3	Color tools with the syntax of <code>colortbl</code>	16
7	The command <code>\RowStyle</code>	16
8	The width of the columns	17
8.1	Basic tools	17
8.2	The columns <code>X</code>	18
9	The exterior rows and columns	18
10	The continuous dotted lines	20
10.1	The option <code>nullify-dots</code>	21
10.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	22
10.3	How to generate the continuous dotted lines transparently	23
10.4	The labels of the dotted lines	24
10.5	Customisation of the dotted lines	24
10.6	The dotted lines and the rules	25

11	The <code>\CodeAfter</code>	25
11.1	The command <code>\line</code> in the <code>\CodeAfter</code>	25
11.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	26
12	The notes in the tabulars	28
12.1	The footnotes	28
12.2	The notes of tabular	28
12.3	Customisation of the tabular notes	30
12.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	32
13	Other features	32
13.1	Use of the column type <code>S</code> of <code>siunitx</code>	32
13.2	Alignment option in <code>{NiceMatrix}</code>	32
13.3	The command <code>\rotate</code>	33
13.4	The option <code>small</code>	33
13.5	The counters <code>iRow</code> and <code>jCol</code>	34
13.6	The option <code>light-syntax</code>	34
13.7	Color of the delimiters	35
13.8	The environment <code>{NiceArrayWithDelims}</code>	35
14	Use of Tikz with <code>nicematrix</code>	35
14.1	The nodes corresponding to the contents of the cells	35
14.2	The “medium nodes” and the “large nodes”	36
14.3	The nodes which indicate the position of the rules	37
14.4	The nodes corresponding to the command <code>\SubMatrix</code>	38
15	API for the developpers	39
16	Technical remarks	40
16.1	Definition of new column types	40
16.2	Diagonal lines	40
16.3	The “empty” cells	41
16.4	The option <code>exterior-arraycolsep</code>	41
16.5	Incompatibilities	41
17	Examples	42
17.1	Utilisation of the key “tikz” of the command <code>\Block</code>	42
17.2	Notes in the tabulars	42
17.3	Dotted lines	43
17.4	Dotted lines which are no longer dotted	44
17.5	Stacks of matrices	45
17.6	How to highlight cells of a matrix	48
17.7	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	50
18	Implementation	51
19	History	216
	Index	223