

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

June 26, 2022

Abstract

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

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

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

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

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

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

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

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

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

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

*This document corresponds to the version 6.10a of `nicematrix`, at the date of 2022/06/26.

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

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

1 The environments of this package

The package `nicematrix` defines the following new environments.

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

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

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

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

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

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

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

2 The vertical space between the rows

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

```


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


```

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

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

The initial value of these parameters is 0 pt in order to have for the environments of `nicematrix` the same behaviour as those of `array` and `amsmath`. However, a value of 1 pt would probably be a good choice and we suggest to set them with `\NiceMatrixOptions`.⁴

```

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


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


```

³In fact, it's possible to use directly the `X` columns in the environment `{NiceTabular}` (and the required width for the tabular is fixed by the key `width`): cf. p. 21

⁴One should remark that these parameters apply also to the columns of type `S` of `siunitx` whereas the package `cellspace` is not able to act on such columns of type `S`.

3 The vertical position of the arrays

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

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

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

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

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

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

1. an item

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

However, it's also possible to use the tools of `booktabs`⁵: `\toprule`, `\bottomrule`, `\midrule`, etc.

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

1. an item

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

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

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

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

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

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

4 The blocks

4.1 General case

In the environments of `nicematrix`, it's possible to use the command `\Block` in order to place an element in the center of a rectangle of merged cells of the array.⁶

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

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

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

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

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

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

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

One may wish to raise the size of the “A” placed in the block of the previous example. Since this element is composed in math mode, it's not possible to use directly a command like `\large`, `\Large` and `\LARGE`. That's why the command `\Block` provides an option between angle brackets to specify some TeX code which will be inserted before the beginning of the math mode.⁷

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

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

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

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

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

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

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

⁷This argument between angular brackets may also be used to insert a command of font such as `\bfseries` when the command `\\` is used in the content of the block.

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;
- the key `fill` takes in as value a color and fills the block with that color;
- the key `draw` takes in as value a color and strokes the frame of the block with that color (the default value of that key is the current color of the rules of the array);
- the key `color` takes in as value a color and apply that color the content of the block but draws also the frame of the block with that color;
- the keys `hlines`, `vlines` and `hvlines` draw all the corresponding rules in the block;
- the key `line-width` is the width of the rules (this key is meaningful only when one of the keys `draw`, `hvlines`, `vlines` and `hlines` is used);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt⁸);
- the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`);
- when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz⁹ by using as options the value of that key `tikz` (which must be a list of keys allowed for a Tikz path). For examples, cf. p. 47;
- the key `name` provides a name to the rectangular Tikz node corresponding to the block; it's possible to use that name with Tikz in the `\CodeAfter` of the environment (cf. p. 28);
- the key `respect-arraystretch` prevents the setting of `\arraystretch` to 1 at the beginning of the block (which is the behaviour by default) ;
- the key `borders` provides the ability to draw only some borders of the blocks; the value of that key is a (comma-separated) list of elements covered by `left`, `right`, `top` and `bottom`; it's possible, in fact, in the list which is the value of the key `borders`, to add an entry of the form `tikz={list}` where `list` is a list of couples `key=value` of Tikz specifying the graphical characteristics of the lines that will be drawn (for an example, see p. 51).

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

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

```
\begin{NiceTabular}{cwc{2cm}wc{3cm}c}
rose      & tulip & daisy & dahlia \\
violet
& \Block[draw=red,fill=[RGB]{204,204,255},rounded-corners]{2-2}
& \LARGE Some beautiful flowers}
& & marigold \\
iris & & lis \\
arum & periwinkle & forget-me-not & hyacinth
\end{NiceTabular}
```

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

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

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

4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

- The natural width of the contents of these blocks is taken into account for the width of the current column.
In the columns with a fixed width (columns `w{...}{...}`, `p{...}`, `b{...}`, `m{...}` and `X`), the content of the block is formatted as a paragraph of that width.
- The specification of the horizontal position provided by the type of column (`c`, `r` or `l`) is taken into account for the blocks.
- The specifications of font specified for the column by a construction `>{...}` in the preamble of the array are taken into account for the mono-column blocks of that column (this behaviour is probably expected).

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

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

4.3 The mono-row blocks

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

4.4 The mono-cell blocks

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

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

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

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

```

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

```

Writer	year of birth
Hugo	1802
Balzac	1799

We recall that if the first mandatory argument of `\Block` is left blank, the block is mono-cell.¹¹

4.5 Horizontal position of the content of the block

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

```

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

```

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

In order to have an horizontal 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.

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

5.1 Some differences with the classical environments

5.1.1 The vertical rules

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

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

First	Second
Peter	
Mary	George

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

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

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

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

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

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

5.1.2 The command `\cline`

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

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

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

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

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

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

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

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

5.2 The thickness and the color of the rules

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

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

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

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

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

5.3 The tools of `nicematrix` for the rules

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

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

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

- These blocks are:
 - the blocks created by the command `\Block`¹² presented p. 4;
 - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `\Vdots`, etc. (cf. p. 23).
- The corners are created by the key `corners` explained below (see p. 10).

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

5.3.1 The keys `hlines` and `vlines`

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

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

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

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

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

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

5.3.2 The keys hvlines and hvlines-except-borders

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

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

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

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

5.3.3 The (empty) corners

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

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

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

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

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

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

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

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

¹⁴For sake of completeness, we should also say that a cell contained in a block (even an empty cell) is not taken into account for the determination of the corners. That behaviour is natural. The precise definition of a “non-empty cell” is given below (cf. p. 46).

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

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

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

▷ The corners are also taken into account by the tools provided by `nicematrix` to color cells, rows and columns. These tools don't color the cells which are in the corners (cf. p. 14).

5.4 The command `\diagbox`

The command `\diagbox` (inspired by the package `diagbox`), allows, when it is used in a cell, to slash that cell diagonally downwards.¹⁵

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

$x \backslash y$	<i>e</i>	<i>a</i>	<i>b</i>	<i>c</i>
<i>e</i>	<i>e</i>	<i>a</i>	<i>b</i>	<i>c</i>
<i>a</i>	<i>a</i>	<i>e</i>	<i>c</i>	<i>b</i>
<i>b</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>a</i>
<i>c</i>	<i>c</i>	<i>b</i>	<i>a</i>	<i>e</i>

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

5.5 Commands for customized rules

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

- the key `command` is the name (without the backslahs) of a command that will be created by `nicematrix` and that will be available for the final user in order to draw horizontal rules (similarly to `\hline`);
- the key `letter` takes in as argument a letter¹⁶ that the user will use in the preamble of an environment with preamble (such as `\NiceTabular`) in order to specify a vertical rule.

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

- *First possibility*

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

- the key `multiplicity` is the number to consecutive rules that will be drawn: for instance, a value of 2 will create double rules such those created by `\hline\hline` or `||` in the preamble of an environment;
- the key `color` sets the color of the rule ;

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

¹⁶The following letters are forbidden: `lcrpmbVX|()[]!@<>`

- the key `sep-color` sets the color between two successive rules (should be used only in conjunction with `multiplicity`).

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

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

- *Second possibility*

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

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

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

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

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

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

one	two	three
four	five	six
seven	eight	nine

- *Third possibility* : the key `dotted`

As one can see, the dots of a dotted line of Tikz have the shape of a square, and not a circle. That's why the extension `nicematrix` provides in the key `custom-line` a key `dotted` which will

draw rounded dots. The initial value of the key `total-width` is, in this case, equal to the diameter of the dots (but the user may change the value with the key `total-width` if needed). Those dotted rules are also used by `nicematrix` to draw continuous dotted rules between cells of the matrix with `\Cdots`, `\Vdots`, etc. (cf. p. 23).

In fact, `nicematrix` defines by default the command `\hdottedline` and the letter “:” for those dotted rules.¹⁷

```
\NiceMatrixOptions % présent dans nicematrix.sty
{
  custom-line =
  {
    letter = : ,
    command = hdottedline ,
    dotted
  }
}
```

Thus, it’s possible to use the command `\hdottedline` to draw a horizontal dotted rule.

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

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

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

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

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

6 The color of the rows and columns

6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

- The package `colortbl` patches `array`, leading to some incompatibilities (for instance with the command `\hdotsfor`).
- The package `colortbl` constructs the array row by row, alternating colored rectangles, rules and contents of the cells. The resulting PDF is difficult to interpret by some PDF viewers and may lead to artefacts on the screen.
 - Some rules seem to disappear. This is because many PDF viewers give priority to graphical element drawn posteriorly (which is in the spirit of the “painting model” of PostScript and PDF). Concerning this problem, MuPDF (which is used, for instance, by SumatraPDF) gives better results than Adobe Reader).

¹⁷However, it’s possible to overwrite those definitions with a `custom-line` (in order, for example, to switch to dashed lines).

- A thin white line may appear between two cells of the same color. This phenomenon occurs when each cell is colored with its own instruction `fill` (the PostScript operator `fill` noted `f` in PDF). This is the case with `colortbl`: each cell is colored on its own, even when `\columncolor` or `\rowcolor` is used.

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

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

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

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

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

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

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

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

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

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

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

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

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

a	b	c
e	f	g
h	i	j

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

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

¹⁹Remark that, in the `\CodeBefore`, PGF/Tikz nodes of the form “(i-j)” are also available to indicate the position to the potential rules: cf. p. 43.

```

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

```

a	b	c
e	f	g
h	i	j

- The command `\arraycolor` takes in as mandatory argument a color and color the whole tabular with that color (excepted the potential exterior rows and columns: cf. p. 21). It's only a particular case of `\rectanglecolor`.
- The command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

```

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

```

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

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

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

```

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

```

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`²⁰. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows

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

of the tabular with the row 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 $i-j$ describes in fact the interval of all the rows of the tabular, beginning with the row i).

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

- The key `cols` describes a set of columns. The command `\rowcolors` will color only the cells of these columns. The value is a comma-separated list of intervals of the form $i-j$ (where i or j may be replaced by $*$).
- With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.²¹
- With the key `respect-blocks` the “rows” alternately colored may extend over several rows if they have to incorporate blocks (created with the command `\Block`: cf. p. 4).

```
\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

- 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.

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


```

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

```

Peter
James
Abigail
Elisabeth
Claudius
Jane
Alexandra

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

```

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

```

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

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

```

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

```

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

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

6.3 Color tools with the syntax of colortbl

It's possible to access the preceding tools with a syntax close to the syntax of `colortbl`. For that, one must use the key `colortbl-like` in the current environment.²² There are three commands available (they are inspired by `colortbl` but are *independent* of `colortbl`):

- `\cellcolor` which colorizes a cell;²³
- `\rowcolor` which must be used in a cell and which colorizes the end of the row;
- `\columncolor` which must be used in the preamble of the environment with the same syntax as the corresponding command of `colortbl` (however, unlike the command `\columncolor` of `colortbl`, this command `\columncolor` can appear within another command, itself used in the preamble of the array).

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

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

7 The command `\RowStyle`

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

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

- The key `nb-rows` sets the number of rows to which the specifications of the current command will apply (with the special value `*`, it will apply to all the following rows).
- The keys `cell-space-top-limit`, `cell-space-bottom-limit` and `cell-space-limits` are available with the same meaning that the corresponding global keys (cf. p. 2).
- The key `rowcolor` sets the color of the background and the key `color` sets the color of the text.²⁴

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

²³However, this command `\cellcolor` will delete the following spaces, which does not the command `\cellcolor` of `colortbl`.

²⁴The key `color` uses the command `\color` but inserts also an instruction `\leavevmode` before. This instruction prevents a extra vertical space in the cells which belong to columns of type `p`, `b`, `m` and `X` (which start in vertical mode).

- The key `bold` enforces bold characters for the cells of the row, both in math and text mode.

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

first	second	third	fourth
1	2	3	4
I	II	III	IV

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

8 The width of the columns

8.1 Basic tools

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

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

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

In the environments of `nicematrix`, it's also possible to fix the *minimal* width of all the columns (excepted the potential exterior columns: cf. p. 21) directly with the key `columns-width`.

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

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

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

It's possible to give the special value `auto` to the option `columns-width`: all the columns of the array will have a width equal to the widest cell of the array.²⁵

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

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

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

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

²⁵The result is achieved with only one compilation (but PGF/Tikz will have written informations in the `aux` file and a message requiring a second compilation will appear).

But it's also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment `{NiceMatrixBlock}` with the option `auto-columns-width`²⁶. The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

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

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

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

8.2 The columns V of varwidth

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

That point is illustrated on the following examples.

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

- first item
- second item

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

- first item
- second item

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

When the package `varwidth` is loaded, the columns `V` of `varwidth` are supported by `nicematrix`. Concerning `nicematrix`, one of the interests of this type of columns is that, for a cell of a column of type `V`, the PGF/Tikz node created by `nicematrix` for the content of that cell has a width adjusted to the content of the cell : cf. p. 41. If the content of the cell is empty, the cell will be considered as empty by `nicematrix` in the construction of the dotted lines and the «empty corners» (that's not the case with a cell of a column `p`, `m` or `b`).

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

²⁶At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

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

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

8.3 The columns X

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

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

For sake of similarity with the environment `{tabularx}`, `nicematrix` also provides an environment `{NiceTabularX}` with a first mandatory argument which is the width of the tabular.²⁷

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

- It's possible to give a weight for the column by providing a positive integer directly as argument of the specifier `X`. For example, a column `X[2]` will have a width double of the width of a column `X` (which has a weight equal to 1).²⁸
- It's possible to specify an horizontal alignment with one of the letters `l`, `c` and `r` (which insert respectively `\raggedright`, `\centering` and `\raggedleft` followed by `\arraybackslash`).
- It's possible to specify a vertical alignment with one of the keys `t` (alias `p`), `m` and `b` (which construct respectively columns of type `p`, `m` and `b`). The default value is `t`.

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

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

9 The exterior rows and columns

The options `first-row`, `last-row`, `first-col` and `last-col` allow the composition of exterior rows and columns in the environments of `nicematrix`. It's particularly interesting for the (mathematical) matrices.

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

²⁷If `tabularx` is loaded, one must use `{NiceTabularX}` (and not `{NiceTabular}`) in order to use the columns `X` (this point comes from a conflict in the definitions of the specifier `X`).

²⁸The negative values of the weight, as provided by `tabu` (which is now obsolete), are *not* supported by `nicematrix`. If such a value is used, an error will be raised.

```

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

```

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

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

We have several remarks to do.

- For the environments with an explicit preamble (i.e. `{NiceTabular}`, `{NiceArray}` and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type `r` for the first column and `l` for the last one.²⁹
- One may wonder how `nicematrix` determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
 - For the environments with explicit preamble, like `{NiceTabular}` and `{pNiceArray}`, the number of columns can obviously be computed from the preamble.
 - When the option `light-syntax` (cf. p. 39) is used, `nicematrix` has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (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

```

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

```

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

```

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

Remarks

- As shown in the previous example, the horizontal and vertical rules don't extend in the exterior rows and columns. This remark also applies to the customized rules created by the key `custom-line` (cf. p. 11).
- A specification of color present in `code-for-first-row` also applies to a dotted line drawn in that exterior “first row” (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 19) doesn't apply to the “first column” and “last column”.
- For technical reasons, it's not possible to use the option of the command `\\` after the “first row” or before the “last row”. The placement of the delimiters would be wrong. If you are looking for a workaround, consider the command `\SubMatrix` in the `\CodeAfter` described p. 29.

10 The continuous dotted lines

Inside the environments of the package `nicematrix`, new commands are defined: `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots`. These commands are intended to be used in place of `\dots`, `\cdots`, `\vdots`, `\ddots` and `\iddots`.³⁰

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells³¹ 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`.³²

```

\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}

```

$$\left[\begin{array}{ccccccc}
a_1 & \cdots & \cdots & \cdots & \cdots & \cdots & a_1 \\
\vdots & & & & & & \\
& & a_2 & \cdots & \cdots & \cdots & a_2 \\
& & \vdots & & & & \\
& & & & \ddots & & \\
& & & & & & \\
a_1 & & a_2 & & & & a_n
\end{array} \right]$$

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}

```

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

³⁰The 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`.

³¹The precise definition of a “non-empty cell” is given below (cf. p. 46).

³²It's also possible to change the color of all these dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 27.

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

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

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

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

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

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\Vdots` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.³³

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

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

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

10.1 The option `nullify-dots`

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

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

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

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

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

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

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

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

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

³³In `nicematrix`, one should use `\hspace*` and not `\hspace` for such an usage because `nicematrix` loads `array`. One may also remark that it's possible to fix the width of a column by using the environment `{NiceArray}` (or one of its variants) with a column of type `w` or `W`: see p. 19

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

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

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

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

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

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

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

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

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

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

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

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`³⁴ is loaded (but you might have problem if you use `\rowcolor` on the same row as `\Hdotsfor`).

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

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

³⁴We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

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

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`.³⁵

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`³⁰ and `\hdotsfor` are redefined within the environments provided by `nicematrix` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` (“automatic dots” of `amsmath`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

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

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

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

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

10.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `\CodeAfter` which is described p. 29) accept two optional arguments specified by the tokens `_` and `^` for labels positionned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

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

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

³⁵The options `renew-dots`, `renew-matrix` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage`.

10.5 Customisation of the dotted lines

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

- `color`;
- `radius`;
- `shorten-start`, `shorten-end` and `shorten`;
- `inter`;
- `line-style`.

These options may also be fixed with `\NiceMatrixOptions`, as options of `\CodeAfter` or at the level of a given environment but, in those cases, they must be prefixed by `xdots` (*xdots* to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.), and, thus have for names:

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

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

The option `xdots/color`

The option `xdots/color` fixes the color of the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 21.

New 6.9 The option `xdots/radius`

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

The option `xdots/shorten`

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

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

New 6.9 The option `xdots/inter`

The option `xdots/inter` fixes the length between the dots. The initial value is 0.45 em (it is recommended to use a unit of length dependent of the current font).

The option `xdots/line-style`

It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).³⁶

```
\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

³⁶The 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.

(and this system uses PGF and not Tikz). This style is called `standard` and that's the initial value of the parameter `xdots/line-style`.

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

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

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

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

10.6 The dotted lines and the rules

The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier `l` in the preamble, by the command `\Hline`, by the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` and by the tools created by `custom-line` are not drawn within the blocks).³⁷

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

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

11 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.³⁸

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

The experienced users may, for instance, use the PGF/Tikz nodes created by `nicematrix` in the `\CodeAfter`. These nodes are described further beginning on p. 40.

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

³⁷On the other side, the command `\line` in the `\CodeAfter` (cf. p. 29) does *not* create block.

³⁸There is also a key `code-before` described p. 14.

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

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

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

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

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

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

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

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. 45).

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

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

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

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

- the first argument is the left delimiter, which may be any extensible delimiter provided by LaTeX : `(`, `[`, `\{`, `\langle`, `\lgroup`, `\lfloor`, etc. but also the null delimiter `.`;
- the second argument is the upper-left corner of the submatrix with the syntax i - j where i the number of row and j the number of column;
- the third argument is the lower-right corner with the same syntax;
- the fourth argument is the right delimiter;
- the last argument, which is optional, is a list of `key=value` pairs.³⁹

³⁹There is no optional argument between square brackets in first position because a square bracket just after `\SubMatrix` must be interpreted as the first (mandatory) argument of the command `\SubMatrix`: that bracket is the left delimiter of the sub-matrix to construct (eg.: `\SubMatrix[{2-2}{4-7}]`).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

It's also possible to specify some delimiters⁴⁰ by placing them in the preamble of the environment (for the environments with a preamble: `{NiceArray}`, `{pNiceArray}`, etc.). This syntax is inspired by the extension `blkarray`.

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

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

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

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

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

- the first argument is the upper-left corner of the submatrix with the syntax $i-j$ where i the number of row and j the number of column;
- the second argument is the lower-right corner with the same syntax;
- the third argument is the label of the brace that will be put by `nicematrix` (with PGF) above the brace (for the command `\OverBrace`) or under the brace (for `\UnderBrace`).

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

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

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

⁴⁰Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

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

```

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

```

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

12 The notes in the tabulars

12.1 The footnotes

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

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

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

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

12.2 The notes of tabular

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

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

```

\begin{NiceTabular}{@{\llr@{}}
\toprule \RowStyle{\bfseries}
Last name & First name & Birth day & \\
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.}
& Jacques & 5 juin 1962 & \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.}
& Mathilde & 23 mai 1988 & \\

```



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

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

^a Achard is an old family of the Poitou.

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

- If you have several successive commands `\tabularnote{...}` with no space at all between them, the labels of the corresponding notes are composed together, separated by commas (this is similar to the option `multiple` of `footmisc` for the footnotes).
 - If a command `\tabularnote{...}` is exactly at the end of a cell (with no space at all after), the label of the note is composed in an overlapping position (towards the right). This structure may provide a better alignment of the cells of a given column.
 - If the key `notes/para` is used, the notes are composed at the end of the array in a single paragraph (as with the key `para` of `threeparttable`).
 - There is a key `tabularnote` which provides a way to insert some text in the zone of the notes before the numbered tabular notes.
 - If the package `booktabs` has been loaded (before or after `nicematrix`), the key `notes/bottomrule` draws a `\bottomrule` of `booktabs` after the notes.
 - The command `\tabularnote` may be used *before* the environment of `nicematrix`. Thus, it's possible to use it on the title inserted by `\caption` in an environment `{table}` of LaTeX.
- New 6.8** If several commands `\tabularnote` are used in a tabular with the same argument, only one note is inserted at the end of the tabular (but all the labels are composed, of course). It's possible to control that feature with the key `notes/detect-duplicates`.
- It's possible to create a reference to a tabular note created by `\tabularnote` (with the usual command `\label` used after the `\tabularnote`).

For an illustration of some of those remarks, see table 1, p. 34. This table has been composed with the following code.

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

```

Touchet & Marie\tabularnote{This note is shared by two references.} & 89 \\
Wallis & John & 87 \\
\bottomrule
\end{NiceTabular}
\end{table}

```

Table 1: Use of `\tabularnote`^a

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

Some text before the notes.

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

^b Considered as the first nurse of history.

^c Nicknamed “the Lady with the Lamp”.

^d This note is shared by two references.

^e The label of the note is overlapping.

12.3 Customisation of the tabular notes

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

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

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

```

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

```

We detail these keys.

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

Initial value: `false`

That key is also available within a given environment.

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

Initial value: `false`

That key is also available within a given environment.

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

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

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

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

Initial value: `#1`

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

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

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

Initial value: `#1`

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

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

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

- The notes are composed at the end of the tabular by using internally a style of list of `enumitem`. This style of list is defined as follows (with, of course, keys of `enumitem`):

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

The specification `align = left` in that style requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 34).

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

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

Initially, the style of list is defined by: `afterlabel = \nobreak, itemjoin = \quad`

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

Initial value: `empty`

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

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

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

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

Initial value : `true`

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

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

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

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

13 Other features

14 Autres fonctionnalités

14.1 Command `\ShowCellNames`

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

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

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

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

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

```


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


```

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

14.3 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).

```


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


```

14.4 The command `\rotate`

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

In the following command, we use that command in the `code-for-first-row`.⁴¹

```

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

```

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

14.5 The option `small`

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

⁴¹It can also be used in `\RowStyle` (cf. p. 18).

```

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

```

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

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

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

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

14.6 The counters `iRow` and `jCol`

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column⁴². Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 14) and in the `\CodeAfter` (cf. p. 28), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

```

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

```

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

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

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).

⁴²We 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.

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

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

14.7 The option `light-syntax`

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

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

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

$$\begin{matrix} & a & b \\ a \left[\begin{matrix} 2 \cos a & \cos a + \cos b \end{matrix} \right. \\ b \left[\begin{matrix} \cos a + \cos b & 2 \cos b \end{matrix} \right. \end{matrix}$$

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

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.⁴³

14.8 Color of the delimiters

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

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

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

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

14.9 The environment `{NiceArrayWithDelims}`

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

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

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

⁴³The reason is that, when the option `light-syntax` is used, the whole content of the environment is loaded as a TeX argument to be analyzed. The environment doesn't behave in that case as a standard environment of LaTeX which only put TeX commands before and after the content.

14.10 The command `\OnlyMainNiceMatrix`

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

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

15 Use of Tikz with `nicematrix`

15.1 The nodes corresponding to the contents of the cells

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

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

However, it's possible to impose the creation of a node with the command `\NotEmpty`.⁴⁴

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

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

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

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

```


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


```

`\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) ;`

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

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

```


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


```

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

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

⁴⁴One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 23) and the computation of the “corners” (cf. p. 10).

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

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

15.1.1 The columns V of varwidth

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

```
\begin{NiceTabular}{V{10cm}}
\bfseries \large
Titre \\\
\lipsum[1][1-4]
\CodeAfter
\tikz \draw [rounded corners] (1-1) -| (last-|2) -- (last-|1) |- (1-1) ;
\end{NiceTabular}
```

Titre

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna.

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

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

In fact, the package `nicematrix` can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option `create-medium-nodes` and the second ones with the option `create-large-nodes`.⁴⁵

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.⁴⁶

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

⁴⁵There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

⁴⁶There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 21).

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.⁴⁷

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

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. 14). It’s possible to have these nodes available in the `\CodeBefore` by using the key `create-cell-nodes` of the keyword `\CodeBefore` (in that case, the nodes are created first before the construction of the array by using informations written on the `aux` file and created a second time during the contruction of the array itself).

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

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

⁴⁷The options `left-margin` and `right-margin` take dimensions as values but, if no value is given, the default value is used, which is `\arraycolsep` (by default: 5 pt). There is also an option `margin` to fix both `left-margin` and `right-margin` to the same value.

```

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

```

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

15.3 The nodes which indicate the position of the rules

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

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

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

```

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

```

```

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

```

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

```

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

```

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

15.4 The nodes corresponding to the command `\SubMatrix`

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

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

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

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

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

16 API for the developers

The package `nicematrix` provides two variables which are internal but public⁴⁸:

- `\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).

⁴⁸According to the LaTeX3 conventions, each variable with name beginning with `\g_nicematrix` ou `\l_nicematrix` is public and each variable with name beginning with `\g__nicematrix` or `\l__nicematrix` is private.

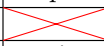
Example : We want to write a command `\crossbox` to draw a cross in the current cell. This command will take in an optional argument between square brackets for a list of pairs *key-value* which will be given to Tikz before the drawing.
It's possible to program such command `\crossbox` as follows, explicitly using the public variable `\g_nicematrix_code_after_tl`.

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

\NewDocumentCommand \crossbox { ! O { } }
{
  \tl_gput_right:Nx \g_nicematrix_code_after_tl
  {
    \__pantigny_crossbox:nnn
    { \int_use:c { c@iRow } }
    { \int_use:c { c@jCol } }
    { \exp_not:n { #1 } }
  }
}
\ExplSyntaxOff
```

Here is an example of utilisation:

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

merlan	requin	cabillaud
baleine		morue
mante	raie	poule

17 Technical remarks

17.1 Diagonal lines

By default, all the diagonal lines⁴⁹ 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):

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

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

⁴⁹We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in the `\CodeAfter`.

```

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

```

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

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

The same example without parallelization:

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

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

17.2 The “empty” cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides. When the key `corners` is used (cf. p. 10), `nicematrix` computes corners consisting of empty cells. However, an “empty cell” is not necessarily a cell with no TeX content (that is to say a cell with no token between the two ampersands `&`). The precise rules are as follow.

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

```

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

```

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

- Each cell whose TeX output has a width equal to zero is empty.
- A cell containing the command `\NotEmpty` is not empty (and a PGF/Tikz node) is created in that cell.
- A cell with a command `\Hspace` (or `\Hspace*`) is empty. This command `\Hspace` is a command defined by the package `nicematrix` with the same meaning as `\hspace` except that the cell where it is used is considered as empty. This command can be used to fix the width of some columns of the matrix without interfering with `nicematrix`.
- A cell of a column of type `p`, `m` or `t` is always considered as not empty. *Caution* : One should not rely upon that point because it may change in a future version of `nicematrix`. On the other side, a cell of a column of type `V` of `varwidth` (cf. p. 20) is empty when its TeX content has a width equal to zero.

17.3 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea⁵⁰. The environment `{matrix}`

⁵⁰In the documentation of `{amsmath}`, we can read: *The extra space of `\arraycolsep` that `array` adds on each side is a waste so we remove it [in `{matrix}`] (perhaps we should instead remove it from `array` in general, but that's a harder task).*

of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`⁵¹. The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` of `array` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

17.4 Incompatibilities

The package `nicematrix` is not compatible with the class `ieeeaccess` (because that class is not compatible with PGF/Tikz).⁵²

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

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

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

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

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

18 Examples

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

The key `tikz` of the command `\Block` is available only when Tikz is loaded.⁵³ For the following example, we need also the Tikz library `patterns`.

```
\usetikzlibrary{patterns}

\ttfamily \small
\begin{NiceTabular}{X[m]X[m]X[m]}[hvlines, cell-space-limits=3pt]
  \Block[tikz={pattern=grid, pattern color=lightgray}]{ }
  {pattern = grid, \ pattern color = lightgray}
& \Block[tikz={pattern = north west lines, pattern color=blue}]{ }
  {pattern = north west lines, \ pattern color = blue}
& \Block[tikz={outer color = red!50, inner color=white }]{2-1}
  {outer color = red!50, \ inner color = white} \
  \Block[tikz={pattern = sixpointed stars, pattern color = blue!15}]{ }
  {pattern = sixpointed stars, \ pattern color = blue!15}
```

⁵¹And 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.

⁵²See <https://tex.stackexchange.com/questions/528975/error-loading-tikz-in-ieeeaccess-class>

⁵³By default, `nicematrix` only loads PGF, which is a sub-layer of Tikz.

```
& \Block[tikz={left color = blue!50}]{  
  {left color = blue!50} \\  
\end{NiceTabular}
```

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

18.2 Notes in the tabulars

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

Let's consider that we wish to number the notes of a tabular with stars.⁵⁴

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument ⁵⁵

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

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

```
\NiceMatrixOptions  
{  
  notes =  
  {  
    style = \stars{#1} ,  
    enumitem-keys =  
    {  
      widest* = \value{tabularnote} ,  
      align = right  
    }  
  }  
}  
  
\begin{NiceTabular}{{}}{llr{}}  
\toprule \RowStyle{\bfseries}  
Last name & First name & Birth day \\  
\midrule  
Achard\tabularnote{Achard is an old family of the Poitou.}  
& Jacques & 5 juin 1962 \\  
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.}  
& Mathilde & 23 mai 1988 \\  

```

⁵⁴Of course, it's realistic only when there is very few notes in the tabular.

⁵⁵In fact: the value of its argument.


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

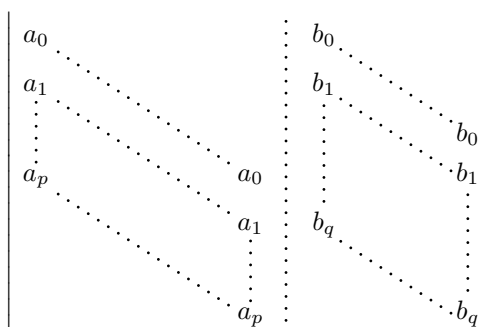
Last name	First name	Birth day
Achard*	Jacques	June 5, 2005
Lefebvre**	Mathilde	January 23, 1975
Vanesse	Stephany	October 30, 1994
Dupont	Chantal	January 15, 1998

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

18.3 Dotted lines

An example with the resultant of two polynoms:

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



An example for a linear system:

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

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

18.4 Dotted lines which are no longer dotted

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

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

In fact, it's even possible to draw solid lines with the commands `\Cdots`, `\Vdots`, etc.⁵⁶

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

⁵⁶In this document, the Tikz library `arrows.meta` has been loaded, which impacts the shape of the arrow tips.

```

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

```

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

18.5 Dashed rules

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

```

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

```

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

18.6 Stacks of matrices

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

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

```

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

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

```

```

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

```

```

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

```

```

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

```

```

\end{NiceMatrixBlock}

```

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

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

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

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

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

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

```

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

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

```

`\end{pNiceArray}$`

...

`\end{NiceMatrixBlock}`

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

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

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

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

18.7 How to highlight cells of a matrix

In order to highlight a cell of a matrix, it's possible to “draw” that cell with the key **draw** of the command `\Block` (this is one of the uses of a mono-cell block⁵⁷).

```

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

```

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

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don't spread the cells of the array. We recall that, on the other side, the commands `\hline` and `\Hline`, the specifier “|” and the options `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` spread the cells.⁵⁸

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

```

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

```

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

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

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

However, it's not possible to do a fine tuning. That's why we describe now a method to highlight a row of the matrix.

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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

```
\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}
\end{tikzpicture}
\end{NiceArray}
```

```

\node [highlight = (9-2) (9-6)] { } ;
\node [highlight = (2-9) (6-9)] { } ;
\end{tikzpicture}
\Body
& & & & & & & \color{blue}\scriptstyle C_j \\
& & & & & & b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\
& & & & & & \Vdots & & \Vdots & & \Vdots \\
& & & & & & & & b_{kj} \\
& & & & & & & & \Vdots \\
& & & & & & b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn} \\
& a_{11} & \Cdots & & & a_{1n} \\
& \Vdots & & & & \Vdots & & & \Vdots \\
\color{blue}\scriptstyle L_i
& a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} & \Cdots & & c_{ij} \\
& \Vdots & & & & \Vdots \\
& a_{n1} & \Cdots & & & a_{nn} \\
\CodeAfter
\tikz \draw [gray,shorten > = 1mm, shorten < = 1mm] (9-4.north) to [bend left] (4-9.west) ;
\end{NiceArray}\}

```

19 Implementation

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

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

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

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

Declaration of the package and packages loaded

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

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

<@@=nicematrix>

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

```

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

```

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

```

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

```

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

We load some packages.

```

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

```

```

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

```

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

```

18 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
19 {
20   \bool_if:NTF \c_@@_messages_for_Overleaf_bool
21     { \msg_new:nnn { nicematrix } { #1 } { #2 } { #3 } }
22     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
23 }
24
25 \cs_new_protected:Npn \@@_msg_redirect_name:nn
26 { \msg_redirect_name:nnn { nicematrix } }

```

Technical definitions

```

27 \tl_new:N \l_@@_argspec_tl
28 \cs_generate_variant:Nn \seq_gset_split:Nnn { N V n }
29 \cs_generate_variant:Nn \keys_define:nn { n x }
30 \hook_gput_code:nnn { begindocument } { . }
31 {
32   \@ifpackageloaded { varwidth }
33     { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_true_bool } }
34     { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_false_bool } }
35   \@ifpackageloaded { arydshln }
36     { \bool_const:Nn \c_@@_arydshln_loaded_bool { \c_true_bool } }
37     { \bool_const:Nn \c_@@_arydshln_loaded_bool { \c_false_bool } }
38   \@ifpackageloaded { booktabs }
39     { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_true_bool } }
40     { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_false_bool } }
41   \@ifpackageloaded { enumitem }
42     { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_true_bool } }
43     { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_false_bool } }
44   \@ifpackageloaded { tabularx }
45     { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_true_bool } }
46     { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_false_bool } }
47   { }
48   \@ifpackageloaded { tikz }
49   {

```

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

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

```

50   \bool_const:Nn \c_@@_tikz_loaded_bool \c_true_bool
51   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
52   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }

```

```

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

```

We test whether the current class is revtex4-1 (deprecated) or revtex4-2 because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date January 2022, the current version revtex4-2 is 4.2e (compatible with `booktabs`).

```

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

```

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

```

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

```

```

71 \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.

```

72 \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.

```

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

```

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.

```

84 \ProvideDocumentCommand \iddots { }
85 {
86     \mathinner
87     {
88         \tex_mkern:D 1 mu
89         \box_move_up:nn { 1 pt } { \hbox:n { . } }
90         \tex_mkern:D 2 mu
91         \box_move_up:nn { 4 pt } { \hbox:n { . } }
92         \tex_mkern:D 2 mu
93         \box_move_up:nn { 7 pt }
94         { \vbox:n { \kern 7 pt \hbox:n { . } } }
95         \tex_mkern:D 1 mu

```

```

96     }
97 }

```

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.

```

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

```

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

```

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

```

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

```

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

```

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.

```

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

```

Idem for `\CT@drs@`.

```

126   \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
127   \cs_set:Npn \CT@drs@ #1 #2
128   {
129     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
130       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
131   }
132   \cs_set:Npn \hline
133   {
134     \noalign { \ifnum 0 = ` } \fi
135     \cs_set_eq:NN \hskip \vskip
136     \cs_set_eq:NN \vrule \hrule
137     \cs_set_eq:NN \@width \@height
138     { \CT@arc@ \vline }
139     \futurelet \reserved@a
140     \@xhline
141   }
142 }
143 }

```

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.

```

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

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`⁵⁹

```

153   \skip_horizontal:N \c_zero_dim
154 }

```

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`.

```

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

```

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.

```

159 \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`.

```

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

```

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

```

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

```

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).

```

172   \int_compare:nNnT { #1 } < { #2 }
173   { \multispan { \int_eval:n { #2 - #1 } } & }
174   \multispan { \int_eval:n { #3 - #2 + 1 } }
175   {
176     \CT@arc@
177     \leaders \hrule \@height \arrayrulewidth \hfill
178     \skip_horizontal:N \c_zero_dim
179   }

```

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

```

180   \peek_meaning_remove_ignore_spaces:NTF \cline
181   { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
182   { \everycr { } \cr }

```

⁵⁹See question 99041 on TeX StackExchange.

```

183 }
184 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

The following command is a small shortcut.
185 \cs_new:Npn \@@_math_toggle_token:
186 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

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

193 \cs_new_protected:Npn \@@_set_CT@drsc@:
194 { \peek_meaning:NTF [ \@@_set_CT@drsc@_i: \@@_set_CT@drsc@_ii: }
195 \cs_new_protected:Npn \@@_set_CT@drsc@_i: [ #1 ] #2 \q_stop
196 { \cs_set:Npn \CT@drsc@ { \color [ #1 ] { #2 } } }
197 \cs_new_protected:Npn \@@_set_CT@drsc@_ii: #1 \q_stop
198 { \cs_set:Npn \CT@drsc@ { \color { #1 } } }

199 \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.

```

200 \bool_new:N \l_@@_siunitx_loaded_bool
201 \hook_gput_code:nnn { begindocument } { . }
202 {
203   \ifpackageloaded { siunitx }
204     { \bool_set_true:N \l_@@_siunitx_loaded_bool }
205     { }
206 }

```

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

```

207 \hook_gput_code:nnn { begindocument } { . }
208 {
209   \bool_if:nTF { ! \l_@@_siunitx_loaded_bool }
210     { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
211     {
212       \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
213       {
214         \renewcommand*{\NC@rewrite@S}[1] []
215         {
216           \@temptokena is a toks (not supported by the L3 programming layer).
217           \@temptokena \exp_after:wN
218           { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
219           \NC@find
220         }
221       }
222     }

```

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.

```

223 \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).

```
224 \cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }
```

The command `\NiceMatrixLastEnv` is not used by the package `nicematrix`. It's only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it's meant to be used after the environment in order to refer to that environment — and its nodes — without having to give it a name). This command *must* be expandable since it will be used in `pgf` nodes.

```
225 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
226 { \int_use:N \g_@@_env_int }
```

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

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

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

```
229 \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).

```
230 \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.).

```
231 \dim_new:N \l_@@_col_width_dim
232 \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.

```
233 \int_new:N \g_@@_row_total_int
234 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

```
237 \str_new:N \l_@@_hpos_cell_str
238 \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`.

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


Idem pour the mono-row blocks.

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

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

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

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

```
243 \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.

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

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

```
245 \bool_new:N \l_@@_notes_detect_duplicates_bool
246 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

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

```
247 \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.

```
248 \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.

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

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

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

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

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

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

```
252 \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.

```
253 \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 contain 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 }`).

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

```

255 \cs_new_protected:Npn \@@_test_if_math_mode:
256 {
257   \if_mode_math: \else:
258     \@@_fatal:n { Outside-math-mode }
259   \fi:
260 }

```

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

```

261 \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.

```

262 \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”.

```

263 \colorlet { nicematrix-last-col } { . }
264 \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*).

```

265 \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*.

```

266 \tl_new:N \g_@@_com_or_env_str
267 \tl_gset:Nn \g_@@_com_or_env_str { environment }

```

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

```

268 \cs_new:Npn \@@_full_name_env:
269 {
270   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
271     { command \space \c_backslash_str \g_@@_name_env_str }
272     { environment \space \{ \g_@@_name_env_str \} }
273 }

```

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

```

274 \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.

```

275 \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.

```

276 \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.

```
277 \int_new:N \l_@@_old_iRow_int
278 \int_new:N \l_@@_old_jCol_int
```

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

The following sequence will contain the names (without backslash) of the commands created by `custom-line` (commands used by the final user in order to draw horizontal rules).

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

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

```
280 \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.

```
281 \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.

```
282 \bool_new:N \l_@@_X_columns_aux_bool
283 \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.

```
284 \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).

```
285 \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).

```
286 \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`).

```
287 \tl_new:N \l_@@_code_before_tl
288 \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).

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

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

```
290 \dim_new:N \l_@@_x_initial_dim
291 \dim_new:N \l_@@_y_initial_dim
292 \dim_new:N \l_@@_x_final_dim
293 \dim_new:N \l_@@_y_final_dim
```

The L3 programming layer provides scratch dimensions `\l_tmpa_dim` and `\l_tmpb_dim`. We creates two more in the same spirit.

```
294 \dim_zero_new:N \l_@@_tmpc_dim
295 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
296 \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”.

```
297 \dim_new:N \g_@@_width_last_col_dim
298 \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.

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

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

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

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

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

```
301 \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.

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

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

```
303 \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`).

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

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

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

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{n}{...}{...}` with $n > 1$ is issued. In `\g_@@_multicolumn_sizes_seq`, the “sizes” (that is to say the values of n) correspondant will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```
306 \seq_new:N \g_@@_multicolumn_cells_seq
307 \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).

```
308 \int_new:N \l_@@_row_min_int
309 \int_new:N \l_@@_row_max_int
310 \int_new:N \l_@@_col_min_int
311 \int_new:N \l_@@_col_max_int
```

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

```
312 \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).

```
313 \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`.

```
314 \tl_new:N \l_@@_fill_tl
315 \tl_new:N \l_@@_draw_tl
316 \seq_new:N \l_@@_tikz_seq
317 \clist_new:N \l_@@_borders_clist
318 \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`.

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

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

```
320 \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`).

```
321 \str_new:N \l_@@_hpos_block_str
322 \str_set:Nn \l_@@_hpos_block_str { c }
323 \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`.

```
324 \tl_new:N \l_@@_vpos_of_block_tl
325 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

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

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

```
327 \bool_new:N \l_@@_vlines_block_bool
328 \bool_new:N \l_@@_hlines_block_bool
```

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

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

330 \dim_new:N \l_@@_submatrix_extra_height_dim
331 \dim_new:N \l_@@_submatrix_left_xshift_dim
332 \dim_new:N \l_@@_submatrix_right_xshift_dim
333 \clist_new:N \l_@@_hlines_clist
334 \clist_new:N \l_@@_vlines_clist
335 \clist_new:N \l_@@_submatrix_hlines_clist
336 \clist_new:N \l_@@_submatrix_vlines_clist
```

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

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

Variables for the exterior rows and columns

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

• First row

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

```
338 \int_new:N \l_@@_first_row_int
339 \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.

```
340 \int_new:N \l_@@_first_col_int
341 \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).

```
342 \int_new:N \l_@@_last_row_int
343 \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”.⁶⁰

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

⁶⁰We 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`

```
345 \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.

```
346 \int_new:N \l_@@_last_col_int
347 \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.

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

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

Some utilities

```
349 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
350 {
351   \tl_set:Nn \l_tmpa_tl { #1 }
352   \tl_set:Nn \l_tmpb_tl { #2 }
353 }
```

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

```
354 \cs_new_protected:Npn \@@_expand_clist:N #1
355 {
356   \clist_if_in:NnF #1 { all }
357   {
358     \clist_clear:N \l_tmpa_clist
359     \clist_map_inline:Nn #1
360     {
361       \tl_if_in:nnTF { ##1 } { - }
362       { \@@_cut_on_hyphen:w ##1 \q_stop }
363       {
364         \tl_set:Nn \l_tmpa_tl { ##1 }
365         \tl_set:Nn \l_tmpb_tl { ##1 }
366       }
367       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
368       { \clist_put_right:Nn \l_tmpa_clist { ###1 } }
369     }
370     \tl_set_eq:NN #1 \l_tmpa_clist
371   }
372 }
```

The command `\tabularnote`

The LaTeX counter `tabularnote` 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.

```
373 \newcounter { tabularnote }
```

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

```
374 \seq_new:N \g_@@_tabularnotes_seq
```

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

```
375 \tl_new:N \l_@@_tabularnote_tl
```

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

```
377 \newcounter{nicematrix_draft}
378 \cs_new_protected:Npn \@@_notes_format:n #1
379 {
380   \setcounter { nicematrix_draft } { #1 }
381   \@@_notes_style:n { nicematrix_draft }
382 }
```

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

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

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

```
384 \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`.

```
385 \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`.

```
386 \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`).

```
387 \hook_gput_code:nnn { begindocument } { . }
388 {
389   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
390   {
391     \NewDocumentCommand \tabularnote { m }
392     { \@@_error:n { enumitem-not-loaded } }
393   }
394 }
```


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.

```

395     \newlist { tabularnotes } { enumerate } { 1 }
396     \setlist [ tabularnotes ]
397     {
398         topsep = Opt ,
399         noitemsep ,
400         leftmargin = * ,
401         align = left ,
402         labelsep = Opt ,
403         label =
404             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
405     }
406     \newlist { tabularnotes* } { enumerate* } { 1 }
407     \setlist [ tabularnotes* ]
408     {
409         afterlabel = \nobreak ,
410         itemjoin = \quad ,
411         label =
412             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
413     }

```

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.⁶¹

```

414     \NewDocumentCommand \tabularnote { m }
415     {
416         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
417         { \@@_error:n { tabularnote~forbidden } }
418         {

```

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

```

419         \int_zero:N \l_tmpa_int
420         \bool_if:NT \l_@@_notes_detect_duplicates_bool
421         {
422             \seq_map_indexed_inline:Nn \g_@@_tabularnotes_seq
423             {
424                 \tl_if_eq:nnT { #1 } { ##2 }
425                 { \int_set:Nn \l_tmpa_int { ##1 } \seq_map_break: }
426             }
427         }
428         \int_compare:nNnTF \l_tmpa_int = 0
429         {
430             \stepcounter { tabularnote }
431             \seq_put_right:Nx \l_@@_notes_labels_seq
432             { \@@_notes_format:n { \int_use:c { c @ tabularnote } } }
433             \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
434         }
435         {
436             \seq_put_right:Nx \l_@@_notes_labels_seq

```

⁶¹We should try to find a solution to that problem.

```

437         { \@@_notes_format:n { \int_use:N \l_tmpa_int } }
438     }
439     \peek_meaning:NF \tabularnote
440     {

```

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.

```

441         \hbox_set:Nn \l_tmpa_box
442     {

```

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

```

443         \@@_notes_label_in_tabular:n
444         {
445             \seq_use:Nnnn
446             \l_@@_notes_labels_seq { , } { , } { , }
447         }
448     }

```

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.

```

449         \addtocounter { tabularnote } { -1 }
450         \refstepcounter { tabularnote }
451         \seq_clear:N \l_@@_notes_labels_seq
452         \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).

```

453         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
454     }
455 }
456 }
457 }
458 }

```

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.

```

459 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
460 {
461     \begin { pgfscope }
462     \pgfset
463     {
464         outer~sep = \c_zero_dim ,
465         inner~sep = \c_zero_dim ,
466         minimum~size = \c_zero_dim
467     }
468     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
469     \pgfnode
470     { rectangle }
471     { center }
472     {
473         \vbox_to_ht:nn
474         { \dim_abs:n { #5 - #3 } }
475         {
476             \vfill

```

```

477         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
478     }
479 }
480 { #1 }
481 { }
482 \end { pgfscope }
483 }

```

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

```

484 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
485 {
486     \begin { pgfscope }
487     \pgfset
488     {
489         outer~sep = \c_zero_dim ,
490         inner~sep = \c_zero_dim ,
491         minimum~size = \c_zero_dim
492     }
493     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
494     \pgfpointdiff { #3 } { #2 }
495     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
496     \pgfnode
497     { rectangle }
498     { center }
499     {
500         \vbox_to_ht:nn
501         { \dim_abs:n \l_tmpb_dim }
502         { \fill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
503     }
504     { #1 }
505     { }
506     \end { pgfscope }
507 }

```

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.

```

508 \bool_new:N \l_@@_colortbl_like_bool

```

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

```

509 \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`).

```

510 \dim_new:N \l_@@_cell_space_top_limit_dim
511 \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.

```

512 \dim_new:N \l_@@_xdots_inter_dim
513 \hook_gput_code:nnn { begindocument } { . }
514 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }

```

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

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

```

515 \dim_new:N \l_@@_xdots_shorten_start_dim
516 \dim_new:N \l_@@_xdots_shorten_end_dim
517 \hook_gput_code:nnn { begindocument } { . }
518 {
519     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
520     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
521 }
```

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

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

```

522 \dim_new:N \l_@@_xdots_radius_dim
523 \hook_gput_code:nnn { begindocument } { . }
524 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }
```

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

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

```

525 \tl_new:N \l_@@_xdots_line_style_tl
526 \tl_const:Nn \c_@@_standard_tl { standard }
527 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```

528 \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).

```

529 \tl_new:N \l_@@_baseline_tl
530 \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`).

```

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

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

```

532 \bool_new:N \l_@@_parallelize_diags_bool
533 \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.

```

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

```

535 \dim_new:N \l_@@_notes_above_space_dim
536 \hook_gput_code:nnn { begindocument } { . }
537 { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

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

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

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

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

```
539 \bool_new:N \l_@@_respect_arraystretch_bool
```

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

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

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

```
541 \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.

```
542 \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”.

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

```
544 \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).

```
545 \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`).

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

```
547 \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`.

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

```
549 \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.

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

```
551 \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 “:”.

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

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

```
553 \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.

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

```
555 \keys_define:nn { NiceMatrix / xdots }
556 {
557   line-style .code:n =
558   {
559     \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.

```
560     { \cs_if_exist_p:N \tikzpicture }
561     { \str_if_eq_p:nn { #1 } { standard } }
562     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
563     { \@@_error:n { bad-option-for~line-style } }
564   } ,
565   line-style .value_required:n = true ,
566   color .tl_set:N = \l_@@_xdots_color_tl ,
567   color .value_required:n = true ,
568   shorten .code:n =
569     \hook_gput_code:nnn { begindocument } { . }
570     {
571       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
572       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
573     } ,
574   shorten-start .code:n =
575     \hook_gput_code:nnn { begindocument } { . }
576     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
577   shorten-end .code:n =
578     \hook_gput_code:nnn { begindocument } { . }
579     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
```

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

```
580   shorten .value_required:n = true ,
581   shorten-start .value_required:n = true ,
582   shorten-end .value_required:n = true ,
583   radius .code:n =
584     \hook_gput_code:nnn { begindocument } { . }
585     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
586   radius .value_required:n = true ,
587   inter .code:n =
588     \hook_gput_code:nnn { begindocument } { . }
589     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
590   radius .value_required:n = true ,
```

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

```
591   down .tl_set:N = \l_@@_xdots_down_tl ,
592   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).

```
593   draw-first .code:n = \prg_do_nothing: ,
594   unknown .code:n = \@@_error:n { Unknown-key-for~xdots }
595 }
```

```

596 \keys_define:nn { NiceMatrix / rules }
597 {
598   color .tl_set:N = \l_@@_rules_color_tl ,
599   color .value_required:n = true ,
600   width .dim_set:N = \arrayrulewidth ,
601   width .value_required:n = true
602 }

```

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

```

603 \keys_define:nn { NiceMatrix / Global }
604 {
605   custom-line .code:n = \@@_custom_line:n { #1 } ,
606   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
607   delimiters .value_required:n = true ,
608   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
609   rules .value_required:n = true ,
610   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
611   standard-cline .default:n = true ,
612   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
613   cell-space-top-limit .value_required:n = true ,
614   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
615   cell-space-bottom-limit .value_required:n = true ,
616   cell-space-limits .meta:n =
617   {
618     cell-space-top-limit = #1 ,
619     cell-space-bottom-limit = #1 ,
620   } ,
621   cell-space-limits .value_required:n = true ,
622   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
623   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
624   light-syntax .default:n = true ,
625   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
626   end-of-row .value_required:n = true ,
627   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
628   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
629   last-row .int_set:N = \l_@@_last_row_int ,
630   last-row .default:n = -1 ,
631   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
632   code-for-first-col .value_required:n = true ,
633   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
634   code-for-last-col .value_required:n = true ,
635   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
636   code-for-first-row .value_required:n = true ,
637   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
638   code-for-last-row .value_required:n = true ,
639   hlines .clist_set:N = \l_@@_hlines_clist ,
640   vlines .clist_set:N = \l_@@_vlines_clist ,
641   hlines .default:n = all ,
642   vlines .default:n = all ,
643   vlines-in-sub-matrix .code:n =
644   {
645     \tl_if_single_token:nTF { #1 }
646     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
647     { \@@_error:n { One~letter~allowed } }
648   } ,
649   vlines-in-sub-matrix .value_required:n = true ,
650   hvlines .code:n =
651   {
652     \clist_set:Nn \l_@@_vlines_clist { all }
653     \clist_set:Nn \l_@@_hlines_clist { all }
654   } ,
655   hvlines-except-borders .code:n =

```

```

656 {
657   \clist_set:Nn \l_@@_vlines_clist { all }
658   \clist_set:Nn \l_@@_hlines_clist { all }
659   \bool_set_true:N \l_@@_except_borders_bool
660 } ,
661 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

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

```

662 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
663 renew-dots .value_forbidden:n = true ,
664 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
665 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
666 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
667 create-extra-nodes .meta:n =
668   { create-medium-nodes , create-large-nodes } ,
669 left-margin .dim_set:N = \l_@@_left_margin_dim ,
670 left-margin .default:n = \arraycolsep ,
671 right-margin .dim_set:N = \l_@@_right_margin_dim ,
672 right-margin .default:n = \arraycolsep ,
673 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
674 margin .default:n = \arraycolsep ,
675 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
676 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
677 extra-margin .meta:n =
678   { extra-left-margin = #1 , extra-right-margin = #1 } ,
679 extra-margin .value_required:n = true ,
680 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
681 respect-arraystretch .default:n = true
682 }

```

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

```

683 \keys_define:nn { NiceMatrix / Env }
684 {

```

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

```

685 hvlines-except-corners .code:n = \@@_fatal:n { hvlines-except-corners } ,
686 hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
687 corners .clist_set:N = \l_@@_corners_clist ,
688 corners .default:n = { NW , SW , NE , SE } ,
689 code-before .code:n =
690 {
691   \tl_if_empty:nF { #1 }
692   {
693     \tl_put_right:Nn \l_@@_code_before_tl { #1 }
694     \bool_set_true:N \l_@@_code_before_bool
695   }
696 } ,

```

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

```

697 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
698 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
699 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
700 baseline .tl_set:N = \l_@@_baseline_tl ,
701 baseline .value_required:n = true ,
702 columns-width .code:n =
703   \tl_if_eq:nnTF { #1 } { auto }
704   { \bool_set_true:N \l_@@_auto_columns_width_bool }
705   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
706 columns-width .value_required:n = true ,
707 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.

```

708 \legacy_if:nF { measuring@ }
709 {
710   \str_set:Nn \l_tmpa_str { #1 }
711   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
712   { \@@_error:nn { Duplicate~name } { #1 } }
713   { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
714   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
715 } ,
716 name .value_required:n = true ,
717 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
718 code-after .value_required:n = true ,
719 colortbl-like .code:n =
720   \bool_set_true:N \l_@@_colortbl_like_bool
721   \bool_set_true:N \l_@@_code_before_bool ,
722 colortbl-like .value_forbidden:n = true
723 }
724 \keys_define:nn { NiceMatrix / notes }
725 {
726   para .bool_set:N = \l_@@_notes_para_bool ,
727   para .default:n = true ,
728   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
729   code-before .value_required:n = true ,
730   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
731   code-after .value_required:n = true ,
732   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
733   bottomrule .default:n = true ,
734   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
735   style .value_required:n = true ,
736   label-in-tabular .code:n =
737     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
738   label-in-tabular .value_required:n = true ,
739   label-in-list .code:n =
740     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
741   label-in-list .value_required:n = true ,
742   enumitem-keys .code:n =
743     {
744       \hook_gput_code:nnn { begindocument } { . }
745       {
746         \bool_if:NT \c_@@_enumitem_loaded_bool
747         { \setlist* [ tabularnotes ] { #1 } }
748       }
749     } ,
750   enumitem-keys .value_required:n = true ,
751   enumitem-keys-para .code:n =
752     {
753       \hook_gput_code:nnn { begindocument } { . }
754       {
755         \bool_if:NT \c_@@_enumitem_loaded_bool
756         { \setlist* [ tabularnotes* ] { #1 } }
757       }
758     } ,
759   enumitem-keys-para .value_required:n = true ,
760   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
761   detect-duplicates .default:n = true ,
762   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
763 }
764 \keys_define:nn { NiceMatrix / delimiters }
765 {
766   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
767   max-width .default:n = true ,
768   color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

769   color .value_required:n = true ,
770 }

```

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

```

771 \keys_define:nn { NiceMatrix }
772 {
773   NiceMatrixOptions .inherit:n =
774     { NiceMatrix / Global } ,
775   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
776   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
777   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
778   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
779   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
780   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
781   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
782   NiceMatrix .inherit:n =
783     {
784       NiceMatrix / Global ,
785       NiceMatrix / Env ,
786     } ,
787   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
788   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
789   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
790   NiceTabular .inherit:n =
791     {
792       NiceMatrix / Global ,
793       NiceMatrix / Env
794     } ,
795   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
796   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
797   NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
798   NiceArray .inherit:n =
799     {
800       NiceMatrix / Global ,
801       NiceMatrix / Env ,
802     } ,
803   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
804   NiceArray / rules .inherit:n = NiceMatrix / rules ,
805   NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
806   pNiceArray .inherit:n =
807     {
808       NiceMatrix / Global ,
809       NiceMatrix / Env ,
810     } ,
811   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
812   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
813   pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
814 }

```

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

```

815 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
816 {
817   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
818   width .value_required:n = true ,
819   last-col .code:n = \tl_if_empty:nF { #1 }
820     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
821     \int_zero:N \l_@@_last_col_int ,
822   small .bool_set:N = \l_@@_small_bool ,
823   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.

```
824   renew-matrix .code:n = \@@_renew_matrix: ,
825   renew-matrix .value_forbidden:n = true ,
```

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

```
826   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.

```
827   columns-width .code:n =
828     \tl_if_eq:nnTF { #1 } { auto }
829     { \@@_error:n { Option~auto~for~columns-width } }
830     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

```
831   allow-duplicate-names .code:n =
832     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
833   allow-duplicate-names .value_forbidden:n = true ,
834   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
835   notes .value_required:n = true ,
836   sub-matrix .code:n =
837     \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
838   sub-matrix .value_required:n = true ,
839   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
840 }
```

`\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.

```
841 \NewDocumentCommand \NiceMatrixOptions { m }
842 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }
```

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

```
843 \keys_define:nn { NiceMatrix / NiceMatrix }
844 {
845   last-col .code:n = \tl_if_empty:nTF {#1}
846     {
847       \bool_set_true:N \l_@@_last_col_without_value_bool
848       \int_set:Nn \l_@@_last_col_int { -1 }
849     }
850     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
851   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
852   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
853   small .bool_set:N = \l_@@_small_bool ,
854   small .value_forbidden:n = true ,
855   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
856 }
```

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

```
857 \keys_define:nn { NiceMatrix / NiceArray }
858 {
```

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.

```

859   small .bool_set:N = \l_@@_small_bool ,
860   small .value_forbidden:n = true ,
861   last-col .code:n = \tl_if_empty:nF { #1 }
862                   { \@@_error:n { last-col~non-empty~for~NiceArray } }
863                   \int_zero:N \l_@@_last_col_int ,
864   notes / para .bool_set:N = \l_@@_notes_para_bool ,
865   notes / para .default:n = true ,
866   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
867   notes / bottomrule .default:n = true ,
868   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
869   tabularnote .value_required:n = true ,
870   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
871   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
872   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
873 }

874 \keys_define:nn { NiceMatrix / pNiceArray }
875 {
876   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
877   last-col .code:n = \tl_if_empty:nF {#1}
878                   { \@@_error:n { last-col~non-empty~for~NiceArray } }
879                   \int_zero:N \l_@@_last_col_int ,
880   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
881   small .bool_set:N = \l_@@_small_bool ,
882   small .value_forbidden:n = true ,
883   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
884   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
885   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
886 }

```

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

```

887 \keys_define:nn { NiceMatrix / NiceTabular }
888 {

```

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

```

889   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
890                   \bool_set_true:N \l_@@_width_used_bool ,
891   width .value_required:n = true ,
892   notes / para .bool_set:N = \l_@@_notes_para_bool ,
893   notes / para .default:n = true ,
894   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
895   notes / bottomrule .default:n = true ,
896   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
897   tabularnote .value_required:n = true ,
898   last-col .code:n = \tl_if_empty:nF {#1}
899                   { \@@_error:n { last-col~non-empty~for~NiceArray } }
900                   \int_zero:N \l_@@_last_col_int ,
901   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
902   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
903   unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
904 }

```

Important code used by `{NiceArrayWithDelims}`

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

```

905 \cs_new_protected:Npn \@@_cell_begin:w
906 {

```

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*).

```

907 \tl_gclear:N \g_@@_post_action_cell_tl

```

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

```

908 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

909 \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.

```

910 \int_compare:nNnT \c@jCol = 1
911 { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }

```

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

```

912 \hbox_set:Nw \l_@@_cell_box
913 \bool_if:NF \l_@@_NiceTabular_bool
914 {
915   \c_math_toggle_token
916   \bool_if:NT \l_@@_small_bool \scriptstyle
917 }

```

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

```

918 \color { nicematrix }
919 \g_@@_row_style_tl

```

We will call *corners* of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn't always exist simultaneously).

The codes `\l_@@_code_for_first_row_tl` and `al` don't apply in the corners of the matrix.

```

920 \int_compare:nNnTF \c@iRow = 0
921 {
922   \int_compare:nNnT \c@jCol > 0
923   {
924     \l_@@_code_for_first_row_tl
925     \xglobal \colorlet { nicematrix-first-row } { . }
926   }
927 }
928 {
929   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
930   {
931     \l_@@_code_for_last_row_tl
932     \xglobal \colorlet { nicematrix-last-row } { . }
933   }
934 }
935 }

```

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.

```

936 \cs_new_protected:Npn \@@_begin_of_row:
937 {
938   \int_gincr:N \c@iRow

```

```

939 \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
940 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
941 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
942 \pgfpicture
943 \pgfrememberpicturepositiononpagetrue
944 \pgfcoordinate
945 { \@@_env: - row - \int_use:N \c@iRow - base }
946 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
947 \str_if_empty:NF \l_@@_name_str
948 {
949   \pgfnodealias
950   { \l_@@_name_str - row - \int_use:N \c@iRow - base }
951   { \@@_env: - row - \int_use:N \c@iRow - base }
952 }
953 \endpgfpicture
954 }

```

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.

```

955 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
956 {
957   \int_compare:nNnTF \c@iRow = 0
958   {
959     \dim_gset:Nn \g_@@_dp_row_zero_dim
960     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
961     \dim_gset:Nn \g_@@_ht_row_zero_dim
962     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
963   }
964   {
965     \int_compare:nNnT \c@iRow = 1
966     {
967       \dim_gset:Nn \g_@@_ht_row_one_dim
968       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
969     }
970   }
971 }
972 \cs_new_protected:Npn \@@_rotate_cell_box:
973 {
974   \box_rotate:Nn \l_@@_cell_box { 90 }
975   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
976   {
977     \vbox_set_top:Nn \l_@@_cell_box
978     {
979       \vbox_to_zero:n { }
980       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
981       \box_use:N \l_@@_cell_box
982     }
983   }
984   \bool_gset_false:N \g_@@_rotate_bool
985 }
986 \cs_new_protected:Npn \@@_adjust_size_box:
987 {
988   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
989   {
990     \box_set_wd:Nn \l_@@_cell_box
991     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
992     \dim_gzero:N \g_@@_blocks_wd_dim
993   }

```

```

994 \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
995 {
996   \box_set_dp:Nn \l_@@_cell_box
997   { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
998   \dim_gzero:N \g_@@_blocks_dp_dim
999 }
1000 \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1001 {
1002   \box_set_ht:Nn \l_@@_cell_box
1003   { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1004   \dim_gzero:N \g_@@_blocks_ht_dim
1005 }
1006 }
1007 \cs_new_protected:Npn \@@_cell_end:
1008 {
1009   \@@_math_toggle_token:
1010   \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.

```

1011 \g_@@_post_action_cell_tl
1012 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1013 \@@_adjust_size_box:
1014 \box_set_ht:Nn \l_@@_cell_box
1015 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1016 \box_set_dp:Nn \l_@@_cell_box
1017 { \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”).

```

1018 \dim_gset:Nn \g_@@_max_cell_width_dim
1019 { \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”.

```

1020 \@@_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.

```

1021 \bool_if:NTF \g_@@_empty_cell_bool
1022 { \box_use_drop:N \l_@@_cell_box }
1023 {
1024   \bool_lazy_or:nnTF
1025   \g_@@_not_empty_cell_bool
1026   { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1027   \@@_node_for_cell:
1028   { \box_use_drop:N \l_@@_cell_box }

```

```

1029     }
1030     \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1031     \bool_gset_false:N \g_@@_empty_cell_bool
1032     \bool_gset_false:N \g_@@_not_empty_cell_bool
1033 }

```

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

```

1034 \cs_new_protected:Npn \@@_node_for_cell:
1035 {
1036     \pgfpicture
1037     \pgfsetbaseline \c_zero_dim
1038     \pgfrememberpicturepositiononpagetrue
1039     \pgfset
1040     {
1041         inner~sep = \c_zero_dim ,
1042         minimum~width = \c_zero_dim
1043     }
1044     \pgfnode
1045     { rectangle }
1046     { base }
1047     { \box_use_drop:N \l_@@_cell_box }
1048     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1049     { }
1050     \str_if_empty:NF \l_@@_name_str
1051     {
1052         \pgfnodealias
1053         { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1054         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1055     }
1056     \endpgfpicture
1057 }

```

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.

```

1058 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1059 {
1060     \cs_new_protected:Npn \@@_patch_node_for_cell:
1061     {
1062         \hbox_set:Nn \l_@@_cell_box
1063         {
1064             \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1065             \hbox_overlap_left:n
1066             {
1067                 \pgfsys@markposition
1068                 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - 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.

```

1069             #1
1070         }
1071         \box_use:N \l_@@_cell_box
1072         \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1073         \hbox_overlap_left:n
1074         {
1075             \pgfsys@markposition
1076             { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1077             #1
1078         }
1079     }
1080 }
1081 }

```


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

```

1082 \bool_lazy_or:nNTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1083 {
1084   \@@_patch_node_for_cell:n
1085     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1086 }
1087 { \@@_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.

```

1088 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1089 {
1090   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1091     { g_@@_ #2 _ lines _ tl }
1092   {
1093     \use:c { @@ _ draw _ #2 : nnn }
1094     { \int_use:N \c@iRow }
1095     { \int_use:N \c@jCol }
1096     { \exp_not:n { #3 } }
1097   }
1098 }

1099 \cs_new_protected:Npn \@@_array:
1100 {
1101   \bool_if:NTF \l_@@_NiceTabular_bool
1102     { \dim_set_eq:NN \col@sep \tabcolsep }
1103     { \dim_set_eq:NN \col@sep \arraycolsep }
1104   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1105     { \cs_set_nopar:Npn \@halignto { } }
1106     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1107   \@tabarray

```

`\l_@@_baseline_tl` may have the value `t`, `c` or `b`. However, if the value is `b`, we compose the `\array` (of array) with the option `t` and the right translation will be done further. Remark that `\str_if_eq:VnTF` is fully expandable and you need something fully expandable here.

```

1108   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1109 }

```

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.

```

1110 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1111 \cs_new_protected:Npn \@@_create_row_node:
1112 {
1113   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1114   {
1115     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1116     \@@_create_row_node_i:
1117   }
1118 }
1119 \cs_new_protected:Npn \@@_create_row_node_i:
1120 {

```

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

```

1121   \hbox
1122   {
1123     \bool_if:NT \l_@@_code_before_bool
1124     {
1125       \vtop
1126       {
1127         \skip_vertical:N 0.5\arrayrulewidth
1128         \pgfsys@markposition
1129         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1130         \skip_vertical:N -0.5\arrayrulewidth
1131       }
1132     }
1133     \pgfpicture
1134     \pgfrememberpicturepositiononpagetrue
1135     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1136     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1137     \str_if_empty:NF \l_@@_name_str
1138     {
1139       \pgfnodealias
1140       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1141       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1142     }
1143     \endpgfpicture
1144   }
1145 }

```

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

```

1146 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1147 \cs_new_protected:Npn \@@_everycr_i:
1148 {
1149   \int_gzero:N \c@jCol
1150   \bool_gset_false:N \g_@@_after_col_zero_bool
1151   \bool_if:NF \g_@@_row_of_col_done_bool
1152   {
1153     \@@_create_row_node:

```

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

```

1154   \tl_if_empty:NF \l_@@_hlines_clist
1155   {
1156     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1157     {
1158       \exp_args:NNx
1159       \clist_if_in:NnT
1160       \l_@@_hlines_clist
1161       { \int_eval:n { \c@iRow + 1 } }
1162     }
1163   }

```

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.

```

1164         \int_compare:nNnT \c@iRow > { -1 }
1165         {
1166             \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@`.

```

1167             { \hrule height \arrayrulewidth width \c_zero_dim }
1168         }
1169     }
1170 }
1171 }
1172 }

```

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`).

```

1173 \cs_set_protected:Npn \@@_newcolumntype #1
1174 {
1175     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1176     \peek_meaning:NTF [
1177         { \newcol@ #1 }
1178         { \newcol@ #1 [ 0 ] }
1179     }

```

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

```

1180 \cs_set_protected:Npn \@@_renew_dots:
1181 {
1182     \cs_set_eq:NN \ldots \@@_Ldots
1183     \cs_set_eq:NN \cdots \@@_Cdots
1184     \cs_set_eq:NN \vdots \@@_Vdots
1185     \cs_set_eq:NN \ddots \@@_Ddots
1186     \cs_set_eq:NN \iddots \@@_Iddots
1187     \cs_set_eq:NN \dots \@@_Ldots
1188     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1189 }

```

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

```

1190 \cs_new_protected:Npn \@@_colortbl_like:
1191 {
1192     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1193     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1194     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1195 }

```

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

```

1196 \cs_new_protected:Npn \@@_pre_array_ii:
1197 {

```

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

```

1198     \xglobal \colorlet { nicematrix } { . }

```

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

```

1199     \int_gzero:N \g_@@_total_X_weight_int
1200     \@@_expand_clist:N \l_@@_hlines_clist
1201     \@@_expand_clist:N \l_@@_vlines_clist

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` *before* the vertical skip (and thus, at a wrong place). That's 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 redefinition⁶².

```

1202 \bool_if:NT \c_@@_booktabs_loaded_bool
1203 { \tl_put_left:Nn \@BTnormal \@_create_row_node_i: } % modified in 6.10a
1204 \box_clear_new:N \l_@@_cell_box
1205 \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}`).

```

1206 \bool_if:NT \l_@@_small_bool
1207 {
1208     \cs_set_nopar:Npn \arraystretch { 0.47 }
1209     \dim_set:Nn \arraycolsep { 1.45 pt }
1210 }

1211 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1212 {
1213     \tl_put_right:Nn \@_begin_of_row:
1214     {
1215         \pgfsys@markposition
1216         { \@_env: - row - \int_use:N \c@iRow - base }
1217     }
1218 }

```

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`.

```

1219 \cs_set_nopar:Npn \ialign
1220 {
1221     \bool_if:NTF \l_@@_colortbl_loaded_bool
1222     {
1223         \CT@everycr
1224         {
1225             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1226             \@_everycr:
1227         }
1228     }
1229     { \everycr { \@_everycr: } }
1230     \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`⁶³ and `\extrarowheight` (of `array`). That box is inserted (via `\@arstrut`) in the beginning of each row of the array. That's why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of `\@arstrutbox` and that's why we do it in the `\ialign`.

```

1231 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1232 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1233 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1234 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }

```

⁶²cf. `\nicematrix@redefine@check@rerun`

⁶³The option `small` of `nicematrix` changes (among others) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1235 \dim_gzero_new:N \g_@@_ht_row_one_dim
1236 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1237 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1238 \dim_gzero_new:N \g_@@_ht_last_row_dim
1239 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1240 \dim_gzero_new:N \g_@@_dp_last_row_dim
1241 \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`.

```

1242 \cs_set_eq:NN \ialign \@@_old_ialign:
1243 \halign
1244 }

```

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).

```

1245 \cs_set_eq:NN \@@_old_ldots \ldots
1246 \cs_set_eq:NN \@@_old_cdots \cdots
1247 \cs_set_eq:NN \@@_old_vdots \vdots
1248 \cs_set_eq:NN \@@_old_ddots \ddots
1249 \cs_set_eq:NN \@@_old_iddots \iddots
1250 \bool_if:NTF \l_@@_standard_cline_bool
1251 { \cs_set_eq:NN \cline \@@_standard_cline }
1252 { \cs_set_eq:NN \cline \@@_cline }
1253 \cs_set_eq:NN \Ldots \@@_Ldots
1254 \cs_set_eq:NN \Cdots \@@_Cdots
1255 \cs_set_eq:NN \Vdots \@@_Vdots
1256 \cs_set_eq:NN \Ddots \@@_Ddots
1257 \cs_set_eq:NN \Iddots \@@_Iddots
1258 \cs_set_eq:NN \Hline \@@_Hline:
1259 \cs_set_eq:NN \Hspace \@@_Hspace:
1260 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1261 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1262 \cs_set_eq:NN \Block \@@_Block:
1263 \cs_set_eq:NN \rotate \@@_rotate:
1264 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1265 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1266 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1267 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1268 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1269 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1270 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1271 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1272 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1273 \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.

```

1274 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1275 \hook_gput_code:nnn { env / tabular / begin } { . }
1276 { \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).

```

1277 \seq_gclear:N \g_@@_multicolumn_cells_seq
1278 \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).

```

1279 \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`).

```
1280 \int_gzero_new:N \g_@@_row_total_int
```

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

```
1281 \int_gzero_new:N \g_@@_col_total_int
```

```
1282 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

```
1283 \@@_renew_NC@rewrite@S:
```

```
1284 \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.

```
1285 \tl_gclear_new:N \g_@@_Cdots_lines_tl
```

```
1286 \tl_gclear_new:N \g_@@_Ldots_lines_tl
```

```
1287 \tl_gclear_new:N \g_@@_Vdots_lines_tl
```

```
1288 \tl_gclear_new:N \g_@@_Ddots_lines_tl
```

```
1289 \tl_gclear_new:N \g_@@_Iddots_lines_tl
```

```
1290 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl
```

```
1291 \tl_gclear_new:N \g_nicematrix_code_before_tl
```

```
1292 }
```

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

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

```
1293 \cs_new_protected:Npn \@@_pre_array:
```

```
1294 {
```

```
1295 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
```

```
1296 \int_gzero_new:N \c@iRow
```

```
1297 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
```

```
1298 \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).

```
1299 \int_compare:nNnT \l_@@_last_row_int = { -1 }
```

```
1300 {
```

```
1301 \bool_set_true:N \l_@@_last_row_without_value_bool
```

```
1302 \bool_if:NT \g_@@_aux_found_bool
```

```
1303 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
```

```
1304 }
```

```
1305 \int_compare:nNnT \l_@@_last_col_int = { -1 }
```

```
1306 {
```

```
1307 \bool_if:NT \g_@@_aux_found_bool
```

```
1308 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
```

```
1309 }
```

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

```
1310 \int_compare:nNnT \l_@@_last_row_int > { -2 }
```

```
1311 {
```

```
1312 \tl_put_right:Nn \@@_update_for_first_and_last_row:
```

```
1313 {
```

```

1314         \dim_gset:Nn \g_@@_ht_last_row_dim
1315         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1316         \dim_gset:Nn \g_@@_dp_last_row_dim
1317         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1318     }
1319 }

```

```

1320 \seq_gclear:N \g_@@_cols_vlism_seq
1321 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1322 \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.

```

1323 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1324 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1325 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1326 \int_gset:Nn \g_@@_last_row_node_int { -1 }

```

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

```

1327 \@@_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.

```

1328 \box_clear_new:N \l_@@_the_array_box

```

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

```

1329 \@@_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]`).

```

1330 \dim_zero_new:N \l_@@_left_delim_dim
1331 \dim_zero_new:N \l_@@_right_delim_dim
1332 \bool_if:NTF \l_@@_NiceArray_bool
1333 {
1334     \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1335     \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1336 }
1337 {

```

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

```

1338 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1339 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1340 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1341 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1342 }

```

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).

```

1343 \hbox_set:Nw \l_@@_the_array_box
1344 \skip_horizontal:N \l_@@_left_margin_dim
1345 \skip_horizontal:N \l_@@_extra_left_margin_dim
1346 \c_math_toggle_token
1347 \bool_if:NTF \l_@@_light_syntax_bool
1348   { \use:c { @@-light-syntax } }
1349   { \use:c { @@-normal-syntax } }
1350 }

```

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

```

1351 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1352 {
1353   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1354   \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...

```

1355 \@@_pre_array:
1356 }

```

The `\CodeBefore`

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

```

1357 \cs_new_protected:Npn \@@_pre_code_before:
1358 {

```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1359 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1360 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1361 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1362 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }

```

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

```

1363 \pgfsys@markposition { \@@_env: - position }
1364 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1365 \pgfpicture
1366 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1367 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1368 {
1369   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1370   \pgfcoordinate { \@@_env: - row - ##1 }
1371   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1372 }

```


Now, the recreation of the col nodes.

```

1373 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1374 {
1375   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1376   \pgfcoordinate { \@@_env: - col - ##1 }
1377   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1378 }

```

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

```

1379 \@@_create_diag_nodes:

```

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

```

1380 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1381 \endpgfpicture

```

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

```

1382 \@@_create_blocks_nodes:
1383 \bool_if:NT \c_@@_tikz_loaded_bool
1384 {
1385   \tikzset
1386   {
1387     every~picture / .style =
1388     { overlay , name~prefix = \@@_env: - }
1389   }
1390 }
1391 \cs_set_eq:NN \cellcolor \@@_cellcolor
1392 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1393 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1394 \cs_set_eq:NN \rowcolor \@@_rowcolor
1395 \cs_set_eq:NN \rowcolors \@@_rowcolors
1396 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1397 \cs_set_eq:NN \arraycolor \@@_arraycolor
1398 \cs_set_eq:NN \columncolor \@@_columncolor
1399 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1400 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1401 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1402 }

```

```

1403 \cs_new_protected:Npn \@@_exec_code_before:
1404 {
1405   \seq_gclear_new:N \g_@@_colors_seq
1406   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1407   \group_begin:

```

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

```

1408 \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`.

```

1409 \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.

```

1410 \@@_actually_color:
1411 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1412 \group_end:

```

```

1413 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1414 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1415 }

1416 \keys_define:nn { NiceMatrix / CodeBefore }
1417 {
1418   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1419   create-cell-nodes .default:n = true ,
1420   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1421   sub-matrix .value_required:n = true ,
1422   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1423   delimiters / color .value_required:n = true ,
1424   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1425 }

1426 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1427 {
1428   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1429   \@@_CodeBefore:w
1430 }

```

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.

```

1431 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1432 {
1433   \bool_if:NT \g_@@_aux_found_bool
1434   {
1435     \@@_pre_code_before:
1436     #1
1437   }
1438 }

```

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.

```

1439 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1440 {
1441   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1442   {
1443     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1444     \pgfcoordinate { \@@_env: - row - ##1 - base }
1445     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1446     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1447     {
1448       \cs_if_exist:cT
1449       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1450       {
1451         \pgfsys@getposition
1452         { \@@_env: - ##1 - #####1 - NW }
1453         \@@_node_position:
1454         \pgfsys@getposition
1455         { \@@_env: - ##1 - #####1 - SE }
1456         \@@_node_position_i:
1457         \@@_pgf_rect_node:nnn
1458         { \@@_env: - ##1 - #####1 }
1459         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1460         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1461       }
1462     }
1463 }

```

```

1464 \int_step_inline:nn \c@iRow
1465 {
1466   \pgfnodealias
1467   { \@@_env: - ##1 - last }
1468   { \@@_env: - ##1 - \int_use:N \c@jCol }
1469 }
1470 \int_step_inline:nn \c@jCol
1471 {
1472   \pgfnodealias
1473   { \@@_env: - last - ##1 }
1474   { \@@_env: - \int_use:N \c@iRow - ##1 }
1475 }
1476 \@@_create_extra_nodes:
1477 }

```

```

1478 \cs_new_protected:Npn \@@_create_blocks_nodes:
1479 {
1480   \pgfpicture
1481   \pgf@relevantforpicturesizefalse
1482   \pgfrememberpicturepositiononpagetrue
1483   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1484   { \@@_create_one_block_node:nnnnn ##1 }
1485   \endpgfpicture
1486 }

```

The following command is called `\@@_create_one_block_node:nnnnn` but, in fact, it creates a node only if the last argument (#5) which is the name of the block, is not empty.⁶⁴

```

1487 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1488 {
1489   \tl_if_empty:nF { #5 }
1490   {
1491     \@@_qpoint:n { col - #2 }
1492     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1493     \@@_qpoint:n { #1 }
1494     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1495     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1496     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1497     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1498     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1499     \@@_pgf_rect_node:nnnnn
1500     { \@@_env: - #5 }
1501     { \dim_use:N \l_tmpa_dim }
1502     { \dim_use:N \l_tmpb_dim }
1503     { \dim_use:N \l_@@_tmpc_dim }
1504     { \dim_use:N \l_@@_tmpd_dim }
1505   }
1506 }

```

```

1507 \cs_new_protected:Npn \@@_patch_for_revtext:
1508 {
1509   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1510   \cs_set_eq:NN \insert@column \insert@column@array
1511   \cs_set_eq:NN \@classx \@classx@array
1512   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1513   \cs_set_eq:NN \@arraycr \@arraycr@array
1514   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1515   \cs_set_eq:NN \array \array@array
1516   \cs_set_eq:NN \@array \@array@array

```

⁶⁴Moreover, there is also in the list `\g_@@_pos_of_blocks_seq` the positions of the dotted lines (created by `\Cdots`, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).

```

1517 \cs_set_eq:NN \@tabular \@tabular@array
1518 \cs_set_eq:NN \@mkpream \@mkpream@array
1519 \cs_set_eq:NN \endarray \endarray@array
1520 \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1521 \cs_set:Npn \endtabular { \endarray $\egroup} % $
1522 }

```

The environment {NiceArrayWithDelims}

```

1523 \NewDocumentEnvironment { NiceArrayWithDelims }
1524 { m m O { } m ! O { } t \CodeBefore }
1525 {
1526 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1527 \@@_provide_pgfsyspdfmark:
1528 \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.

```

1529 \bgroup

1530 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1531 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1532 \tl_gset:Nn \g_@@_preamble_tl { #4 }

1533 \int_gzero:N \g_@@_block_box_int
1534 \dim_zero:N \g_@@_width_last_col_dim
1535 \dim_zero:N \g_@@_width_first_col_dim
1536 \bool_gset_false:N \g_@@_row_of_col_done_bool
1537 \str_if_empty:NT \g_@@_name_env_str
1538 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1539 \bool_if:NTF \l_@@_NiceTabular_bool
1540 \mode_leave_vertical:
1541 \@@_test_if_math_mode:
1542 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1543 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁶⁵. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by `colortbl`. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```

1544 \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.

```

1545 \cs_if_exist:NT \tikz@library@external@loaded
1546 {
1547 \tikzexternaldisable
1548 \cs_if_exist:NT \ifstandalone
1549 { \tikzset { external / optimize = false } }
1550 }

```

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

```

1551 \int_gincr:N \g_@@_env_int
1552 \bool_if:NF \l_@@_block_auto_columns_width_bool
1553 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

⁶⁵e.g. `\color[rgb]{0.5,0.5,0}`

The sequence `\g_@@_blocks_seq` will contain the carateristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks (except the blocks with the key `hvlines`).

```
1554 \seq_gclear:N \g_@@_blocks_seq
1555 \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`.

```
1556 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1557 \seq_gclear:N \g_@@_pos_of_xdots_seq
1558 \tl_gclear_new:N \g_@@_code_before_tl
1559 \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.

```
1560 \bool_gset_false:N \g_@@_aux_found_bool
1561 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1562 {
1563   \bool_gset_true:N \g_@@_aux_found_bool
1564   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1565 }
```

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.

```
1566 \tl_gclear:N \g_@@_aux_tl
1567 \tl_if_empty:NF \g_@@_code_before_tl
1568 {
1569   \bool_set_true:N \l_@@_code_before_bool
1570   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1571 }
```

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`.

```
1572 \bool_if:NTF \l_@@_NiceArray_bool
1573 { \keys_set:nn { NiceMatrix / NiceArray } }
1574 { \keys_set:nn { NiceMatrix / pNiceArray } }
1575 { #3 , #5 }

1576 \tl_if_empty:NF \l_@@_rules_color_tl
1577 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
```

The argument `#6` is the last argument of `{NiceArrayWithDelims}`. With that argument of type “`t \CodeBefore`”, we test whether there is the keyword `\CodeBefore` at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword `\CodeBefore` and the (other) keyword `\Body`. It’s the job that will do the command `\@@_CodeBefore_Body:w`. After that job, the command `\@@_CodeBefore_Body:w` will go on with `\@@_pre_array:`.

```
1578 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1579 }
1580 {
1581   \bool_if:NTF \l_@@_light_syntax_bool
1582   { \use:c { end @@-light-syntax } }
1583   { \use:c { end @@-normal-syntax } }
1584   \c_math_toggle_token
1585   \skip_horizontal:N \l_@@_right_margin_dim
1586   \skip_horizontal:N \l_@@_extra_right_margin_dim
1587   \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.

```
1588 \bool_if:NT \l_@@_width_used_bool
```

```

1589 {
1590   \int_compare:nNnT \g_@@_total_X_weight_int = 0
1591     { \@@_error:n { width~without~X~columns } }
1592 }

```

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 .

```

1593 \int_compare:nNnT \g_@@_total_X_weight_int > 0
1594 {
1595   \tl_gput_right:Nx \g_@@_aux_tl
1596   {
1597     \bool_set_true:N \l_@@_X_columns_aux_bool
1598     \dim_set:Nn \l_@@_X_columns_dim
1599     {
1600       \dim_compare:nNnTF
1601         {
1602           \dim_abs:n
1603             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1604         }
1605         <
1606         { 0.001 pt }
1607         { \dim_use:N \l_@@_X_columns_dim }
1608         {
1609           \dim_eval:n
1610             {
1611               ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1612               / \int_use:N \g_@@_total_X_weight_int
1613               + \l_@@_X_columns_dim
1614             }
1615         }
1616     }
1617   }
1618 }

```

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).

```

1619 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1620 {
1621   \bool_if:NF \l_@@_last_row_without_value_bool
1622   {
1623     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1624     {
1625       \@@_error:n { Wrong~last~row }
1626       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1627     }
1628   }
1629 }

```

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”.⁶⁶

```

1630 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1631 \bool_if:nTF \g_@@_last_col_found_bool
1632 { \int_gdecr:N \c@jCol }
1633 {
1634   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1635   { \@@_error:n { last~col~not~used } }
1636 }

```

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

⁶⁶We remind that the potential “first column” (exterior) has the number 0.

```

1637 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1638 \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. 130).

```

1639 \int_compare:nNnT \l_@@_first_col_int = 0
1640 {
1641   \skip_horizontal:N \col@sep
1642   \skip_horizontal:N \g_@@_width_first_col_dim
1643 }

```

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.

```

1644 \bool_if:NTF \l_@@_NiceArray_bool
1645 {
1646   \str_case:VnF \l_@@_baseline_tl
1647   {
1648     b \@@_use_arraybox_with_notes_b:
1649     c \@@_use_arraybox_with_notes_c:
1650   }
1651   \@@_use_arraybox_with_notes:
1652 }

```

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).

```

1653 {
1654   \int_compare:nNnTF \l_@@_first_row_int = 0
1655   {
1656     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1657     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1658   }
1659   { \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”.⁶⁷

```

1660 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1661 {
1662   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1663   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1664 }
1665 { \dim_zero:N \l_tmpb_dim }
1666 \hbox_set:Nn \l_tmpa_box
1667 {
1668   \c_math_toggle_token
1669   \tl_if_empty:NF \l_@@_delimiters_color_tl
1670   { \color { \l_@@_delimiters_color_tl } }
1671   \exp_after:wN \left \g_@@_left_delim_tl
1672   \vcenter
1673   {

```

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. There was a bug in the following line (corrected the 2021/11/23).

```

1674 \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1675 \hbox
1676 {
1677   \bool_if:NTF \l_@@_NiceTabular_bool
1678   { \skip_horizontal:N -\tabcolsep }
1679   { \skip_horizontal:N -\arraycolsep }

```

⁶⁷A 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).

```

1680         \@@_use_arraybox_with_notes_c:
1681         \bool_if:NTF \l_@@_NiceTabular_bool
1682             { \skip_horizontal:N -\tabcolsep }
1683             { \skip_horizontal:N -\arraycolsep }
1684     }

```

We take into account the “last row” (we have previously computed its total height in `\l_tmpb_dim`). There was a bug in the following line (corrected the 2021/11/23).

```

1685         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1686     }

```

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.

```

1687         \tl_if_empty:NF \l_@@_delimiters_color_tl
1688             { \color { \l_@@_delimiters_color_tl } }
1689         \exp_after:wN \right \g_@@_right_delim_tl
1690         \c_math_toggle_token
1691     }

```

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.

```

1692         \bool_if:NTF \l_@@_delimiters_max_width_bool
1693         {
1694             \@@_put_box_in_flow_bis:nn
1695             \g_@@_left_delim_tl \g_@@_right_delim_tl
1696         }
1697         \@@_put_box_in_flow:
1698     }

```

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. 131).

```

1699         \bool_if:NT \g_@@_last_col_found_bool
1700         {
1701             \skip_horizontal:N \g_@@_width_last_col_dim
1702             \skip_horizontal:N \col@sep
1703         }
1704         \bool_if:NF \l_@@_Matrix_bool
1705         {
1706             \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1707             { \@@_error:n { columns-not-used } }
1708         }
1709         \group_begin:
1710         \globaldefs = 1
1711         \@@_msg_redirect_name:nn { columns-not-used } { error }
1712         \group_end:
1713         \@@_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.

```

1714     \egroup

```

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

```

1715         \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1716         \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1717         \iow_now:Nx \@mainaux
1718         {
1719             \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
1720             { \exp_not:V \g_@@_aux_tl }
1721         }
1722         \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1723         \bool_if:NT \c_@@_footnote_bool \endsavenotes
1724     }

```

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.

```
1725 \cs_new_protected:Npn \@@_construct_preamble:
1726 {
```

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 to use the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```
1727 \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).

```
1728 \bool_if:NF \l_@@_Matrix_bool
1729 {
1730   \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1731   \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
```

If the package `varwidth` has defined the column type `V`, we protect from expansion by redefining it to `\@@_V:` (which will be caught by our system).

```
1732 \cs_if_exist:NT \NC@find@V { \@@_newcolumntype V { \@@_V: } }
```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by the L3 programming layer).

```
1733 \exp_args:NV \@temptokena \g_@@_preamble_tl
```

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

```
1734 \@tempswatrue
```

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

```
1735 \@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`.

```
1736 \int_gzero:N \c@jCol
1737 \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1738 \bool_gset_false:N \g_tmpb_bool
1739 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1740 {
1741   \tl_gset:Nn \g_@@_preamble_tl
1742     { ! { \skip_horizontal:N \arrayrulewidth } }
1743 }
1744 {
1745   \clist_if_in:NnT \l_@@_vlines_clist 1
1746   {
```

```

1747         \tl_gset:Nn \g_@@_preamble_tl
1748         { ! { \skip_horizontal:N \arrayrulewidth } }
1749     }
1750 }

```

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`).

```

1751     \seq_clear:N \g_@@_cols_vlism_seq

```

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

```

1752     \int_zero:N \l_tmpa_int

```

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

```

1753     \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1754     \int_gset_eq:NN \g_@@_static_num_of_col_int \c_jCol
1755 }

```

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

```

1756     \bool_if:NT \l_@@_colortbl_like_bool
1757     {
1758         \regex_replace_all:NnN
1759         \c_@@_columncolor_regex
1760         { \c { @@_columncolor_preamble } }
1761         \g_@@_preamble_tl
1762     }

```

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

```

1763     \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}`.

```

1764     \bool_lazy_or:nnT
1765     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1766     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1767     { \bool_set_false:N \l_@@_NiceArray_bool }

```

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

```

1768     \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).

```

1769     \int_compare:nNnTF \l_@@_first_col_int = 0
1770     { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1771     {
1772         \bool_lazy_all:nT
1773         {
1774             \l_@@_NiceArray_bool
1775             { \bool_not_p:n \l_@@_NiceTabular_bool }
1776             { \tl_if_empty_p:N \l_@@_vlines_clist }
1777             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1778         }
1779         { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1780     }
1781     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1782     { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1783     {
1784         \bool_lazy_all:nT
1785         {
1786             \l_@@_NiceArray_bool
1787             { \bool_not_p:n \l_@@_NiceTabular_bool }
1788             { \tl_if_empty_p:N \l_@@_vlines_clist }
1789             { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1790         }
1791         { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1792     }

```

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`).

```

1793   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1794   {
1795     \tl_gput_right:Nn \g_@@_preamble_tl
1796     { > { \@@_error_too_much_cols: } 1 }
1797   }
1798 }

```

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

```

1799 \cs_new_protected:Npn \@@_patch_preamble:n #1
1800 {
1801   \str_case:nnF { #1 }
1802   {
1803     c { \@@_patch_preamble_i:n #1 }
1804     l { \@@_patch_preamble_i:n #1 }
1805     r { \@@_patch_preamble_i:n #1 }
1806     > { \@@_patch_preamble_ii:nn #1 }
1807     ! { \@@_patch_preamble_ii:nn #1 }
1808     @ { \@@_patch_preamble_ii:nn #1 }
1809     | { \@@_patch_preamble_iii:n #1 }
1810     p { \@@_patch_preamble_iv:n #1 }
1811     b { \@@_patch_preamble_iv:n #1 }
1812     m { \@@_patch_preamble_iv:n #1 }
1813     \@@_V: { \@@_patch_preamble_v:n }
1814     V { \@@_patch_preamble_v:n }
1815     \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
1816     \@@_W: { \@@_patch_preamble_vi:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1817     \@@_S: { \@@_patch_preamble_vii:n }
1818     ( { \@@_patch_preamble_viii:nn #1 }
1819     [ { \@@_patch_preamble_viii:nn #1 }
1820     \{ { \@@_patch_preamble_viii:nn #1 }
1821     ) { \@@_patch_preamble_ix:nn #1 }
1822     ] { \@@_patch_preamble_ix:nn #1 }
1823     \} { \@@_patch_preamble_ix:nn #1 }
1824     X { \@@_patch_preamble_x:n }

```

When `tabularx` is loaded, a local redefinition of the specifier `X` is done to replace `X` by `\@@_X`. Thus, our column type `X` will be used in the `{NiceTabularX}`.

```

1825   \@@_X { \@@_patch_preamble_x:n }
1826   \q_stop { }
1827 }
1828 {
1829   \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
1830   {
1831     \seq_gput_right:Nx \g_@@_cols_vlism_seq
1832     { \int_eval:n { \c@jCol + 1 } }
1833     \tl_gput_right:Nx \g_@@_preamble_tl
1834     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1835     \@@_patch_preamble:n
1836   }

```

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

```

1837   {
1838     \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }

```

```

1839         {
1840             \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
1841             \@@_patch_preamble:n
1842         }
1843         { \@@_fatal:nn { unknown~column~type } { #1 } }
1844     }
1845 }
1846 }

```

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

For c, l and r

```

1847 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1848 {
1849     \tl_gput_right:Nn \g_@@_preamble_tl
1850     {
1851         > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
1852         #1
1853         < \@@_cell_end:
1854     }

```

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

```

1855     \int_gincr:N \c@jCol
1856     \@@_patch_preamble_xi:n
1857 }

```

For >, ! and @

```

1858 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1859 {
1860     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1861     \@@_patch_preamble:n
1862 }

```

For |

```

1863 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1864 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1865     \int_incr:N \l_tmpa_int
1866     \@@_patch_preamble_iii_i:n
1867 }
1868 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1869 {
1870     \str_if_eq:nnTF { #1 } |
1871     { \@@_patch_preamble_iii:n | }
1872     {
1873         \tl_gput_right:Nx \g_@@_preamble_tl
1874         {
1875             \exp_not:N !
1876             {
1877                 \skip_horizontal:n
1878                 {

```

Here, the command \dim_eval:n is mandatory.

```

1879                 \dim_eval:n
1880                 {
1881                     \arrayrulewidth * \l_tmpa_int
1882                     + \doublerulesep * ( \l_tmpa_int - 1 )
1883                 }
1884             }
1885         }
1886     }
1887     \tl_gput_right:Nx \g_@@_internal_code_after_tl

```

```

1888     {
1889         \@@_vline:n
1890         {
1891             position = \int_eval:n { \c@jCol + 1 } ,
1892             multiplicity = \int_use:N \l_tmpa_int ,
1893         }

```

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

```

1894     }
1895     \int_zero:N \l_tmpa_int
1896     \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
1897     \@@_patch_preamble:n #1
1898 }
1899 }
1900 \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.

```

1901 \keys_define:nn { WithArrows / p-column }
1902 {
1903     r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
1904     r .value_forbidden:n = true ,
1905     c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
1906     c .value_forbidden:n = true ,
1907     l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
1908     l .value_forbidden:n = true ,
1909     si .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
1910     si .value_forbidden:n = true ,
1911     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
1912     p .value_forbidden:n = true ,
1913     t .meta:n = p ,
1914     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
1915     m .value_forbidden:n = true ,
1916     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
1917     b .value_forbidden:n = true ,
1918 }

```

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

```

1919 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
1920 {
1921     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

1922     \@@_patch_preamble_iv_i:n
1923 }

1924 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
1925 {
1926     \str_if_eq:nnTF { #1 } { [ }
1927     { \@@_patch_preamble_iv_ii:w [ }
1928     { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
1929 }

1930 \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
1931 { \@@_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.

```

1932 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
1933 {

```

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).

```

1934   \str_set:Nn \l_@@_hpos_col_str { j }
1935   \keys_set:nn { WithArrows / p-column } { #1 }
1936   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
1937 }

```

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

```

1938 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
1939 {
1940   \use:x
1941   {
1942     \@@_patch_preamble_iv_v:nnnnnnnn
1943     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
1944     { \dim_eval:n { #1 } }
1945   }

```

The parameter `\l_@@_hpos_col_str` (as `\l_@@_vpos_col_str`) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter `\l_@@_hpos_cell_str` which will provide the horizontal alignment of the column to which belongs the cell.

```

1946   \str_if_eq:VnTF \l_@@_hpos_col_str j
1947   { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
1948   {
1949     \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
1950     { \l_@@_hpos_col_str }
1951   }
1952   \str_case:Vn \l_@@_hpos_col_str
1953   {
1954     c { \exp_not:N \centering }
1955     l { \exp_not:N \raggedright }
1956     r { \exp_not:N \raggedleft }
1957   }
1958 }
1959 { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
1960 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
1961 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
1962 { #2 }
1963 {
1964   \str_case:VnF \l_@@_hpos_col_str
1965   {
1966     { j } { c }
1967     { si } { c }
1968   }
1969   { \l_@@_hpos_col_str }
1970 }
1971 }

```

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

```

1972   \int_gincr:N \c@jCol
1973   \@@_patch_preamble_xi:n
1974 }

```

#1 is the optional argument of `{minipage}` (or `{varwidth}`): `t` of `b`. Indeed, for the columns of type `m`, we use the value `b` here because there is a special post-action in order to center vertically the box (see #4).

#2 is the width of the `{minipage}` (or `{varwidth}`), that is to say also the width of the column.

#3 is the coding for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing). It's also possible to put in that **#3** some code to fix the value of `\l_@@_hpos_cell_str` which will be available in each cell of the column.

#4 is an extra-code which contains `\@@_center_cell_box:` (when the column is a `m` column) or nothing (in the other cases).

#5 is a code put just before the c (or r or l: see #8).
 #6 is a code put just after the c (or r or l: see #8).
 #7 is the type of environment: minipage or varwidth.
 #8 is the lettre c or r or l which is the basic specifier of column which is used *in fine*.

```

1975 \cs_new_protected:Npn \@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
1976 {
1977   \tl_gput_right:Nn \g_@@_preamble_tl
1978   {
1979     > {

```

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.

```

1980       \dim_set:Nn \l_@@_col_width_dim { #2 }
1981       \@_cell_begin:w
1982       \begin { #7 } [ #1 ] { #2 }

```

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

```

1983       \everypar
1984       {
1985         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
1986         \everypar { }
1987       }

```

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

```

1988       #3

```

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

```

1989       \g_@@_row_style_tl
1990       \arraybackslash
1991       #5
1992     }
1993     #8
1994     < {
1995       #6

```

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

```

1996       \@finalstrut \@arstrutbox
1997       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
1998       \end { #7 }

```

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

```

1999       #4
2000       \@_cell_end:
2001     }
2002   }
2003 }

```

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.

```

2004 \cs_new_protected:Npn \@_center_cell_box:
2005 {

```

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

```

2006   \tl_gput_right:Nn \g_@@_post_action_cell_tl
2007   {
2008     \int_compare:nNnT
2009     { \box_ht:N \l_@@_cell_box }
2010     >
2011     { \box_ht:N \@arstrutbox }

```

```

2012     {
2013         \hbox_set:Nn \l_@@_cell_box
2014         {
2015             \box_move_down:nn
2016             {
2017                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2018                   + \baselineskip ) / 2
2019             }
2020             { \box_use:N \l_@@_cell_box }
2021         }
2022     }
2023 }
2024 }

```

For V (similar to the V of varwidth).

```

2025 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2026 {
2027     \str_if_eq:nnTF { #1 } { [ ] }
2028     { \@@_patch_preamble_v_i:w [ ] }
2029     { \@@_patch_preamble_v_i:w [ ] { #1 } }
2030 }
2031 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2032 { \@@_patch_preamble_v_ii:nn { #1 } }
2033 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2034 {
2035     \str_set:Nn \l_@@_vpos_col_str { p }
2036     \str_set:Nn \l_@@_hpos_col_str { j }
2037     \keys_set:nn { WithArrows / p-column } { #1 }
2038     \bool_if:NTF \c_@@_varwidth_loaded_bool
2039     { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2040     {
2041         \@@_error:n { varwidth~not~loaded }
2042         \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2043     }
2044 }

```

For w and W

```

2045 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2046 {
2047     \tl_gput_right:Nn \g_@@_preamble_tl
2048     {
2049         > {

```

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.

```

2050         \dim_set:Nn \l_@@_col_width_dim { #4 }
2051         \hbox_set:Nw \l_@@_cell_box
2052         \@@_cell_begin:w
2053         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2054     }
2055     c
2056     < {
2057         \@@_cell_end:
2058         #1
2059         \hbox_set_end:
2060         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2061         \@@_adjust_size_box:
2062         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2063     }
2064 }

```

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

```

2065     \int_gincr:N \c@jCol
2066     \@@_patch_preamble_xi:n
2067 }

```


For `\@@_S:`. If the user has used `S[...]`, `S` has been replaced by `\@@_S:` during the first expansion of the preamble (done with the tools of standard LaTeX and array).

```

2068 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2069 {
2070   \str_if_eq:nnTF { #1 } { [ ]
2071     { \@@_patch_preamble_vii_i:w [ ]
2072       { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2073     }
2074   \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2075     { \@@_patch_preamble_vii_ii:n { #1 } }
2076   \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2077     {

```

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

```

2078   \cs_if_exist:NTF \siunitx_cell_begin:w
2079   {
2080     \tl_gput_right:Nn \g_@@_preamble_tl
2081     {
2082       > {
2083         \@@_cell_begin:w
2084         \keys_set:nn { siunitx } { #1 }
2085         \siunitx_cell_begin:w
2086       }
2087       c
2088       < { \siunitx_cell_end: \@@_cell_end: }
2089     }

```

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

```

2090     \int_gincr:N \c@jCol
2091     \@@_patch_preamble_xi:n
2092   }
2093   { \@@_fatal:n { Version-of-siunitx-too-old } }
2094 }

```

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

```

2095 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2096 {
2097   \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.

```

2098   \int_compare:nNnTF \c@jCol = \c_zero_int
2099   {
2100     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2101     {

```

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

```

2102       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2103       \tl_gset:Nn \g_@@_right_delim_tl { . }
2104       \@@_patch_preamble:n #2
2105     }
2106     {
2107       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2108       \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2109     }
2110   }
2111   { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2112 }
2113 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2114 {
2115   \tl_gput_right:Nx \g_@@_internal_code_after_tl
2116   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }

```

```

2117 \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
2118 {
2119   \@@_error:nn { delimiter~after~opening } { #2 }
2120   \@@_patch_preamble:n
2121 }
2122 { \@@_patch_preamble:n #2 }
2123 }

```

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}`).

```

2124 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2125 {
2126   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2127   \tl_if_in:nnTF { ) ] \} } { #2 }
2128   { \@@_patch_preamble_ix_i:nnn #1 #2 }
2129   {
2130     \tl_if_eq:nnTF { \q_stop } { #2 }
2131     {
2132       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2133       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2134       {
2135         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2136         \tl_gput_right:Nx \g_@@_internal_code_after_tl
2137         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2138         \@@_patch_preamble:n #2
2139       }
2140     }
2141     {
2142       \tl_if_in:nnT { ( [ \{ } } { #2 }
2143       { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2144       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2145       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2146       \@@_patch_preamble:n #2
2147     }
2148   }
2149 }
2150 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2151 {
2152   \tl_if_eq:nnTF { \q_stop } { #3 }
2153   {
2154     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2155     {
2156       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2157       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2158       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2159       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2160     }
2161     {
2162       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2163       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2164       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2165       \@@_error:nn { double~closing~delimiter } { #2 }
2166     }
2167   }
2168   {
2169     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2170     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2171     \@@_error:nn { double~closing~delimiter } { #2 }
2172     \@@_patch_preamble:n #3
2173   }
2174 }

```

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.

```

2175 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2176 {
2177   \str_if_eq:nnTF { #1 } { [ ]
2178     { \@@_patch_preamble_x_i:w [ ]
2179       { \@@_patch_preamble_x_i:w [ ] #1 }
2180     }
2181   \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2182     { \@@_patch_preamble_x_ii:n { #1 } }

```

#1 is the optional argument of the X specifier (a list of *key-value* pairs).

The following set of keys is for the specifier X in the preamble of the array. Such specifier may have as keys all the keys of { WithArrows / p-column } but also a key as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter \l_@@_weight_int).

```

2183 \keys_define:nn { WithArrows / X-column }
2184 { 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.

```

2185 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2186 {

```

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).

```

2187   \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).

```

2188   \tl_set:Nn \l_@@_vpos_col_str { p }

```

The integer \l_@@_weight_int will be the weight of the X column (the initial value is 1). The user may specify a different value (such as 2, 3, etc.) by putting that value in the optional argument of the specifier. The weights of the X columns are used in the computation of the actual width of those columns as in tabu of tabularray.

```

2189   \int_zero_new:N \l_@@_weight_int
2190   \int_set:Nn \l_@@_weight_int { 1 }
2191   \keys_set:known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
2192   \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2193   \int_compare:nNnT \l_@@_weight_int < 0
2194   {
2195     \@@_error:nx { negative-weight } { \int_use:N \l_@@_weight_int }
2196     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2197   }
2198   \int_gadd:Nn \g_@@_total_X_weight_int \l_@@_weight_int

```

We test whether we know the width of the X-columns by reading the aux file (after the first compilation, the width of the X-columns is computed and written in the aux file).

```

2199   \bool_if:NTF \l_@@_X_columns_aux_bool
2200   {
2201     \@@_patch_preamble_iv_iv:nn
2202     { \l_@@_weight_int \l_@@_X_columns_dim }
2203     { minipage }
2204   }
2205   {
2206     \tl_gput_right:Nn \g_@@_preamble_tl
2207     {
2208       > {
2209         \@@_cell_begin:w
2210         \bool_set_true:N \l_@@_X_column_bool

```

The following code will nullify the box of the cell.

```

2211     \tl_gput_right:Nn \g_@@_post_action_cell_tl
2212     { \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`.

```

2213         \begin { minipage } { 5 cm } \arraybackslash
2214     }
2215     c
2216     < {
2217         \end { minipage }
2218         \@@_cell_end:
2219     }
2220 }
2221 \int_gincr:N \c@jCol
2222 \@@_patch_preamble_xi:n
2223 }
2224 }
```

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.

```

2225 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2226 {
2227     \str_if_eq:nnTF { #1 } { < }
2228     \@@_patch_preamble_xiii:n
2229     {
2230         \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2231         {
2232             \tl_gput_right:Nn \g_@@_preamble_tl
2233             { ! { \skip_horizontal:N \arrayrulewidth } }
2234         }
2235         {
2236             \exp_args:NNx
2237             \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2238             {
2239                 \tl_gput_right:Nn \g_@@_preamble_tl
2240                 { ! { \skip_horizontal:N \arrayrulewidth } }
2241             }
2242         }
2243         \@@_patch_preamble:n { #1 }
2244     }
2245 }
2246 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2247 {
2248     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2249     \@@_patch_preamble_xi:n
2250 }
```

The redefinition of `\multicolumn`

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

```

2251 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2252 {
```

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 that one column specifier in the preamble of `\multicolumn`.

```

2253     \multispan { #1 }
2254     \begingroup
2255     \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
```

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

```

2256 \temptokena = { #2 }
2257 \tempswatrue
2258 \@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.

```

2259 \tl_gclear:N \g_@@_preamble_tl
2260 \exp_after:wN \@@_patch_m_preamble:n \the \temptokena \q_stop

```

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

```

2261 \exp_args:NV \mkpream \g_@@_preamble_tl
2262 \addtopreamble \empty
2263 \endgroup

```

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

```

2264 \int_compare:nNnT { #1 } > 1
2265 {
2266   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2267   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2268   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2269   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2270   {
2271     {
2272       \int_compare:nNnTF \c@jCol = 0
2273       { \int_eval:n { \c@iRow + 1 } }
2274       { \int_use:N \c@iRow }
2275     }
2276     { \int_eval:n { \c@jCol + 1 } }
2277     {
2278       \int_compare:nNnTF \c@jCol = 0
2279       { \int_eval:n { \c@iRow + 1 } }
2280       { \int_use:N \c@iRow }
2281     }
2282     { \int_eval:n { \c@jCol + #1 } }
2283     { } % for the name of the block
2284   }
2285 }

```

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

```

2286 \cs_set:Npn \@sharp { #3 }
2287 \@arstrut
2288 \@preamble
2289 \null

```

We add some lines.

```

2290 \int_gadd:Nn \c@jCol { #1 - 1 }
2291 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2292 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2293 \ignorespaces
2294 }

```

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`.

```

2295 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2296 {
2297   \str_case:nnF { #1 }
2298   {
2299     c { \@@_patch_m_preamble_i:n #1 }
2300     l { \@@_patch_m_preamble_i:n #1 }
2301     r { \@@_patch_m_preamble_i:n #1 }
2302     > { \@@_patch_m_preamble_ii:nn #1 }

```

```

2303     ! { \@@_patch_m_preamble_ii:nn #1 }
2304     @ { \@@_patch_m_preamble_ii:nn #1 }
2305     | { \@@_patch_m_preamble_iii:n #1 }
2306     p { \@@_patch_m_preamble_iv:nnn t #1 }
2307     m { \@@_patch_m_preamble_iv:nnn c #1 }
2308     b { \@@_patch_m_preamble_iv:nnn b #1 }
2309     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2310     \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2311     \q_stop { }
2312   }
2313   { \@@_fatal:nn { unknown~column~type } { #1 } }
2314 }

```

For c, l and r

```

2315 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2316 {
2317   \tl_gput_right:Nn \g_@@_preamble_tl
2318   {
2319     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2320     #1
2321     < \@@_cell_end:
2322   }

```

We test for the presence of a <.

```

2323   \@@_patch_m_preamble_x:n
2324 }

```

For >, ! and @

```

2325 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2326 {
2327   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2328   \@@_patch_m_preamble:n
2329 }

```

For |

```

2330 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2331 {
2332   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2333   \@@_patch_m_preamble:n
2334 }

```

For p, m and b

```

2335 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2336 {
2337   \tl_gput_right:Nn \g_@@_preamble_tl
2338   {
2339     > {
2340       \@@_cell_begin:w
2341       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2342       \mode_leave_vertical:
2343       \arraybackslash
2344       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2345     }
2346     c
2347     < {
2348       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2349       \end { minipage }
2350       \@@_cell_end:
2351     }
2352   }

```

We test for the presence of a <.

```

2353   \@@_patch_m_preamble_x:n
2354 }

```

For w and W

```

2355 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2356 {
2357   \tl_gput_right:Nn \g_@@_preamble_tl
2358   {
2359     > {
2360       \hbox_set:Nw \l_@@_cell_box
2361       \@@_cell_begin:w
2362       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2363     }
2364     c
2365     < {
2366       \@@_cell_end:
2367       #1
2368       \hbox_set_end:
2369       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2370       \@@_adjust_size_box:
2371       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2372     }
2373   }

```

We test for the presence of a <.

```

2374   \@@_patch_m_preamble_x:n
2375 }

```

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.

```

2376 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2377 {
2378   \str_if_eq:nnTF { #1 } { < }
2379   \@@_patch_m_preamble_ix:n
2380   {
2381     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2382     {
2383       \tl_gput_right:Nn \g_@@_preamble_tl
2384       { ! { \skip_horizontal:N \arrayrulewidth } }
2385     }
2386     {
2387       \exp_args:NNx
2388       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2389       {
2390         \tl_gput_right:Nn \g_@@_preamble_tl
2391         { ! { \skip_horizontal:N \arrayrulewidth } }
2392       }
2393     }
2394     \@@_patch_m_preamble:n { #1 }
2395   }
2396 }
2397 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2398 {
2399   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2400   \@@_patch_m_preamble_x:n
2401 }

```

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).

```

2402 \cs_new_protected:Npn \@@_put_box_in_flow:
2403 {
2404   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2405   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2406   \tl_if_eq:NnTF \l_@@_baseline_tl { c }

```

```

2407 { \box_use_drop:N \l_tmpa_box }
2408 \@@_put_box_in_flow_i:
2409 }

```

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).

```

2410 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2411 {
2412   \pgfpicture
2413   \@@_qpoint:n { row - 1 }
2414   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2415   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2416   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2417   \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).

```

2418   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2419   {
2420     \int_set:Nn \l_tmpa_int
2421     {
2422       \str_range:Nnn
2423       \l_@@_baseline_tl
2424       6
2425       { \tl_count:V \l_@@_baseline_tl }
2426     }
2427     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2428   }
2429   {
2430     \str_case:VnF \l_@@_baseline_tl
2431     {
2432       { t } { \int_set:Nn \l_tmpa_int 1 }
2433       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2434     }
2435     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2436     \bool_lazy_or:nnT
2437     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2438     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2439     {
2440       \@@_error:n { bad~value~for~baseline }
2441       \int_set:Nn \l_tmpa_int 1
2442     }
2443     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2444     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2445   }
2446   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2447   \endpgfpicture
2448   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2449   \box_use_drop:N \l_tmpa_box
2450 }

```

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).

```

2451 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2452 {

```

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.


```

2453 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2454 {
2455     \box_set_wd:Nn \l_@@_the_array_box
2456     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2457 }

```

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

```

2458 \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.

```

2459 \hbox
2460 {
2461     \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.

```

2462 \@@_create_extra_nodes:
2463 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2464 }

```

We don't do the following test with `\c@tabularnote` because the value of that counter is not reliable when the command `\ttabbox` of `floatrow` is used (because `\ttabbox` de-activate `\stepcounter` because if compiles several twice its tabular).

```

2465 \bool_lazy_or:nnT
2466 { ! \seq_if_empty_p:N \g_@@_tabularnotes_seq }
2467 { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2468 \@@_insert_tabularnotes:
2469 \end { minipage }
2470 }
2471 \cs_new_protected:Npn \@@_insert_tabularnotes:
2472 {
2473     \skip_vertical:N 0.65ex

```

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

```

2474 \group_begin:
2475 \l_@@_notes_code_before_tl
2476 \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.

```

2477 \int_compare:nNnT \c@tabularnote > 0
2478 {
2479     \bool_if:NTF \l_@@_notes_para_bool
2480     {
2481         \begin { tabularnotes* }
2482         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2483         \end { tabularnotes* }

```

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

```

2484 \par
2485 }
2486 {
2487     \tabularnotes
2488     \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2489     \endtabularnotes
2490 }
2491 }
2492 \unskip
2493 \group_end:
2494 \bool_if:NT \l_@@_notes_bottomrule_bool

```

```

2495 {
2496   \bool_if:NTF \c_@@_booktabs_loaded_bool
2497   {

```

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

```

2498   \skip_vertical:N \aboverulesep

```

`\CT@arc@` is the specification of color defined by `colortbl` but you use it even if `colortbl` is not loaded.

```

2499   { \CT@arc@ \hrule height \heavyrulewidth }
2500   }
2501   { \@@_error:n { bottomrule~without~booktabs } }
2502   }
2503   \l_@@_notes_code_after_tl
2504   \seq_gclear:N \g_@@_tabularnotes_seq
2505   \int_gzero:N \c@tabularnote
2506   }

```

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`.

```

2507 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2508 {
2509   \pgfpicture
2510     \@@_qpoint:n { row - 1 }
2511     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2512     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2513     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2514   \endpgfpicture
2515   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2516   \int_compare:nNnT \l_@@_first_row_int = 0
2517   {
2518     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2519     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2520   }
2521   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2522 }

```

Now, the general case.

```

2523 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2524 {

```

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

```

2525   \tl_if_eq:NnT \l_@@_baseline_tl { t }
2526   { \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`.

```

2527   \pgfpicture
2528   \@@_qpoint:n { row - 1 }
2529   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2530   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2531   {
2532     \int_set:Nn \l_tmpa_int
2533     {
2534       \str_range:Nnn
2535         \l_@@_baseline_tl
2536         6
2537       { \tl_count:V \l_@@_baseline_tl }
2538     }
2539     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2540   }
2541   {
2542     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2543     \bool_lazy_or:nnT
2544       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }

```

```

2545     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2546     {
2547         \@@_error:n { bad-value-for-baseline }
2548         \int_set:Nn \l_tmpa_int 1
2549     }
2550     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2551 }
2552 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2553 \endpgfpicture
2554 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2555 \int_compare:nNnT \l_@@_first_row_int = 0
2556 {
2557     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2558     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2559 }
2560 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2561 }

```

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.

```

2562 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2563 {

```

We will compute the real width of both delimiters used.

```

2564     \dim_zero_new:N \l_@@_real_left_delim_dim
2565     \dim_zero_new:N \l_@@_real_right_delim_dim
2566     \hbox_set:Nn \l_tmpb_box
2567     {
2568         \c_math_toggle_token
2569         \left #1
2570         \vcenter
2571         {
2572             \vbox_to_ht:nn
2573             { \box_ht_plus_dp:N \l_tmpa_box }
2574             { }
2575         }
2576         \right .
2577         \c_math_toggle_token
2578     }
2579     \dim_set:Nn \l_@@_real_left_delim_dim
2580     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2581     \hbox_set:Nn \l_tmpb_box
2582     {
2583         \c_math_toggle_token
2584         \left .
2585         \vbox_to_ht:nn
2586         { \box_ht_plus_dp:N \l_tmpa_box }
2587         { }
2588         \right #2
2589         \c_math_toggle_token
2590     }
2591     \dim_set:Nn \l_@@_real_right_delim_dim
2592     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2593     \skip_horizontal:N \l_@@_left_delim_dim
2594     \skip_horizontal:N -\l_@@_real_left_delim_dim
2595     \@@_put_box_in_flow:
2596     \skip_horizontal:N \l_@@_right_delim_dim
2597     \skip_horizontal:N -\l_@@_real_right_delim_dim
2598 }

```

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).

```
2599 \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).

```
2600 {
2601   \peek_remove_spaces:n
2602   {
2603     \peek_meaning:NTF \end
2604     \@@_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`).

```
2605     { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2606   }
2607 }
2608 {
2609   \@@_create_col_nodes:
2610   \endarray
2611 }
```

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`).

```
2612 \NewDocumentEnvironment { @@-light-syntax } { b }
2613 {
```

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`.

```
2614   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2615   \tl_map_inline:nn { #1 }
2616   {
2617     \str_if_eq:nnT { ##1 } { & }
2618     { \@@_fatal:n { ampersand-in-light-syntax } }
2619     \str_if_eq:nnT { ##1 } { \ }
2620     { \@@_fatal:n { double-backslash-in-light-syntax } }
2621   }
```

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`.

```
2622   \@@_light_syntax_i #1 \CodeAfter \q_stop
2623 }
```

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`.

```
2624 { }
2625 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2626 {
2627   \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).

```
2628   \seq_gclear_new:N \g_@@_rows_seq
2629   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2630   \seq_gset_split:NVn \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.

```
2631 \int_compare:nNnT \l_@@_last_row_int = { -1 }
2632 { \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`).

```
2633 \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.

```
2634 \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2635 \@@_line_with_light_syntax_i:V \l_tmpa_tl
2636 \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2637 \@@_create_col_nodes:
2638 \endarray
2639 }

2640 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2641 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }

2642 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2643 {
2644   \seq_gclear_new:N \g_@@_cells_seq
2645   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2646   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2647   \l_tmpa_tl
2648   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2649 }
2650 \cs_generate_variant:Nn \@@_line_with_light_syntax_i:n { V }
```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security).

```
2651 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2652 {
2653   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2654   { \@@_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.

```
2655 \end { #2 }
2656 }
```

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).

```
2657 \cs_new:Npn \@@_create_col_nodes:
2658 {
2659   \crrc
2660   \int_compare:nNnT \l_@@_first_col_int = 0
2661   {
2662     \omit
2663     \hbox_overlap_left:n
2664     {
2665       \bool_if:NT \l_@@_code_before_bool
2666       { \pgfsys@markposition { \@@_env: - col - 0 } }
2667       \pgfpicture
2668       \pgfrememberpicturepositiononpagetrue
2669       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
2670       \str_if_empty:NF \l_@@_name_str
2671       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2672       \endpgfpicture
```

```

2673         \skip_horizontal:N 2\col@sep
2674         \skip_horizontal:N \g_@@_width_first_col_dim
2675     }
2676     &
2677 }
2678 \omit

```

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

```

2679 \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`).

```

2680 \int_compare:nNnTF \l_@@_first_col_int = 0
2681 {
2682     \bool_if:NT \l_@@_code_before_bool
2683     {
2684         \hbox
2685         {
2686             \skip_horizontal:N -0.5\arrayrulewidth
2687             \pgfsys@markposition { \@@_env: - col - 1 }
2688             \skip_horizontal:N 0.5\arrayrulewidth
2689         }
2690     }
2691     \pgfpicture
2692     \pgfrememberpicturerepositiononpagetrue
2693     \pgfcoordinate { \@@_env: - col - 1 }
2694     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2695     \str_if_empty:NF \l_@@_name_str
2696     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2697     \endpgfpicture
2698 }
2699 {
2700     \bool_if:NT \l_@@_code_before_bool
2701     {
2702         \hbox
2703         {
2704             \skip_horizontal:N 0.5\arrayrulewidth
2705             \pgfsys@markposition { \@@_env: - col - 1 }
2706             \skip_horizontal:N -0.5\arrayrulewidth
2707         }
2708     }
2709     \pgfpicture
2710     \pgfrememberpicturerepositiononpagetrue
2711     \pgfcoordinate { \@@_env: - col - 1 }
2712     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
2713     \str_if_empty:NF \l_@@_name_str
2714     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2715     \endpgfpicture
2716 }

```

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.

```

2717 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2718 \bool_if:NF \l_@@_auto_columns_width_bool
2719 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2720 {
2721     \bool_lazy_and:nnTF
2722     \l_@@_auto_columns_width_bool
2723     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2724     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }

```

```

2725         { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2726     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2727 }
2728 \skip_horizontal:N \g_tmpa_skip
2729 \hbox
2730 {
2731     \bool_if:NT \l_@@_code_before_bool
2732     {
2733         \hbox
2734         {
2735             \skip_horizontal:N -0.5\arrayrulewidth
2736             \pgfsys@markposition { \@@_env: - col - 2 }
2737             \skip_horizontal:N 0.5\arrayrulewidth
2738         }
2739     }
2740     \pgfpicture
2741     \pgfrememberpicturepositiononpagetrue
2742     \pgfcoordinate { \@@_env: - col - 2 }
2743     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2744     \str_if_empty:NF \l_@@_name_str
2745     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2746     \endpgfpicture
2747 }

```

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.

```

2748 \int_gset:Nn \g_tmpa_int 1
2749 \bool_if:NTF \g_@@_last_col_found_bool
2750 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2751 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2752 {
2753     &
2754     \omit
2755     \int_gincr:N \g_tmpa_int

```

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

```

2756     \skip_horizontal:N \g_tmpa_skip
2757     \bool_if:NT \l_@@_code_before_bool
2758     {
2759         \hbox
2760         {
2761             \skip_horizontal:N -0.5\arrayrulewidth
2762             \pgfsys@markposition
2763             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2764             \skip_horizontal:N 0.5\arrayrulewidth
2765         }
2766     }

```

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

```

2767     \pgfpicture
2768     \pgfrememberpicturepositiononpagetrue
2769     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2770     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2771     \str_if_empty:NF \l_@@_name_str
2772     {
2773         \pgfnodealias
2774         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
2775         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2776     }
2777     \endpgfpicture
2778 }

2779 &
2780 \omit

```

The two following lines have been added on 2021-12-15 to solve a bug mentionned by Joao Luis Soares by mail.

```

2781 \int_compare:nNt \g_@@_col_total_int = 1
2782 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
2783 \skip_horizontal:N \g_tmpa_skip
2784 \int_gincr:N \g_tmpa_int
2785 \bool_lazy_all:nT
2786 {
2787   \l_@@_NiceArray_bool
2788   { \bool_not_p:n \l_@@_NiceTabular_bool }
2789   { \clist_if_empty_p:N \l_@@_vlines_clist }
2790   { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2791   { ! \l_@@_bar_at_end_of_pream_bool }
2792 }
2793 { \skip_horizontal:N -\col@sep }
2794 \bool_if:NT \l_@@_code_before_bool
2795 {
2796   \hbox
2797   {
2798     \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.

```

2799 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2800 { \skip_horizontal:N -\arraycolsep }
2801 \pgfsys@markposition
2802 { \@@_env: - col - \int_eval:n {
2803   \g_tmpa_int + 1 } }
2804 \skip_horizontal:N 0.5\arrayrulewidth
2805 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2806 { \skip_horizontal:N \arraycolsep }
2807 }
2808 }
2809 \pgfpicture
2810 \pgfrememberpicturepositiononpagetrue
2811 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2812 {
2813   \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
2814   {
2815     \pgfpoint
2816     { - 0.5 \arrayrulewidth - \arraycolsep }
2817     \c_zero_dim
2818   }
2819   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2820 }
2821 \str_if_empty:NF \l_@@_name_str
2822 {
2823   \pgfnodealias
2824   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
2825   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2826 }
2827 \endpgfpicture

2828 \bool_if:NT \g_@@_last_col_found_bool
2829 {
2830   \hbox_overlap_right:n
2831   {
2832     \skip_horizontal:N \g_@@_width_last_col_dim
2833     \bool_if:NT \l_@@_code_before_bool
2834     {
2835       \pgfsys@markposition
2836       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }

```



```

2837     }
2838     \pgfpicture
2839     \pgfrememberpicturepositiononpagetrue
2840     \pgfcoordinate
2841     { \l_@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2842     \pgfpointorigin
2843     \str_if_empty:NF \l_@@_name_str
2844     {
2845         \pgfnodealias
2846         {
2847             \l_@@_name_str - col
2848             - \int_eval:n { \g_@@_col_total_int + 1 }
2849         }
2850         { \l_@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2851     }
2852     \endpgfpicture
2853 }
2854 }
2855 \cr
2856 }

```

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

```

2857 \tl_const:Nn \c_@@_preamble_first_col_tl
2858 {
2859     >
2860     {

```

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

```

2861     \cs_set_eq:NN \CodeAfter \l_@@_CodeAfter_i:
2862     \bool_gset_true:N \g_@@_after_col_zero_bool
2863     \l_@@_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.

```

2864     \hbox_set:Nw \l_@@_cell_box
2865     \l_@@_math_toggle_token:
2866     \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”.

```

2867     \bool_lazy_and:nnT
2868     { \int_compare_p:nNn \c@iRow > 0 }
2869     {
2870         \bool_lazy_or_p:nn
2871         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2872         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2873     }
2874     {
2875         \l_@@_code_for_first_col_tl
2876         \xglobal \colorlet { nicematrix-first-col } { . }
2877     }
2878 }

```

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`.

```

2879     l
2880     <
2881     {
2882         \l_@@_math_toggle_token:
2883         \hbox_set_end:
2884         \bool_if:NT \g_@@_rotate_bool \l_@@_rotate_cell_box:
2885         \l_@@_adjust_size_box:
2886         \l_@@_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.

```
2887 \dim_gset:Nn \g_@@_width_first_col_dim
2888 { \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.

```
2889 \hbox_overlap_left:n
2890 {
2891   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2892     \@@_node_for_cell:
2893     { \box_use_drop:N \l_@@_cell_box }
2894     \skip_horizontal:N \l_@@_left_delim_dim
2895     \skip_horizontal:N \l_@@_left_margin_dim
2896     \skip_horizontal:N \l_@@_extra_left_margin_dim
2897   }
2898   \bool_gset_false:N \g_@@_empty_cell_bool
2899   \skip_horizontal:N -2\col@sep
2900 }
2901 }
```

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

```
2902 \tl_const:Nn \c_@@_preamble_last_col_tl
2903 {
2904   >
2905   {
```

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

```
2906 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```
2907 \bool_gset_true:N \g_@@_last_col_found_bool
2908 \int_gincr:N \c@jCol
2909 \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.

```
2910 \hbox_set:Nw \l_@@_cell_box
2911 \@@_math_toggle_token:
2912 \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”.

```
2913 \int_compare:nNnT \c@iRow > 0
2914 {
2915   \bool_lazy_or:nnT
2916     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2917     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2918     {
2919       \l_@@_code_for_last_col_tl
2920       \xglobal \colorlet { nicematrix-last-col } { . }
2921     }
2922   }
2923 }
2924 1
2925 <
2926 {
2927   \@@_math_toggle_token:
2928   \hbox_set_end:
2929   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2930   \@@_adjust_size_box:
2931   \@@_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.

```

2932 \dim_gset:Nn \g_@@_width_last_col_dim
2933 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2934 \skip_horizontal:N -2\col@sep

```

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

```

2935 \hbox_overlap_right:n
2936 {
2937   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2938   {
2939     \skip_horizontal:N \l_@@_right_delim_dim
2940     \skip_horizontal:N \l_@@_right_margin_dim
2941     \skip_horizontal:N \l_@@_extra_right_margin_dim
2942     \@@_node_for_cell:
2943   }
2944 }
2945 \bool_gset_false:N \g_@@_empty_cell_bool
2946 }
2947 }

```

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.

```

2948 \NewDocumentEnvironment { NiceArray } { }
2949 {
2950   \bool_set_true:N \l_@@_NiceArray_bool
2951   \str_if_empty:NT \g_@@_name_env_str
2952   { \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).

```

2953   \NiceArrayWithDelims . .
2954 }
2955 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

2956 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2957 {
2958   \NewDocumentEnvironment { #1 NiceArray } { }
2959   {
2960     \str_if_empty:NT \g_@@_name_env_str
2961     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2962     \@@_test_if_math_mode:
2963     \NiceArrayWithDelims #2 #3
2964   }
2965   { \endNiceArrayWithDelims }
2966 }
2967 \@@_def_env:nnn p ( )
2968 \@@_def_env:nnn b [ ]
2969 \@@_def_env:nnn B \{ \}
2970 \@@_def_env:nnn v | |
2971 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2972 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2973 {
2974   \bool_set_true:N \l_@@_Matrix_bool
2975   \use:c { #1 NiceArray }
2976   {
2977     *
2978     {
2979       \int_compare:nNnTF \l_@@_last_col_int < 0
2980       \c@MaxMatrixCols
2981       { \int_eval:n { \l_@@_last_col_int - 1 } }
2982     }
2983     { > \@@_cell_begin:w #2 < \@@_cell_end: }
2984   }
2985 }
2986 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n e }
2987 \clist_map_inline:nn { { } , p , b , B , v , V }
2988 {
2989   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2990   {
2991     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2992     \tl_set:Nn \l_@@_type_of_col_tl c
2993     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2994     \@@_begin_of_NiceMatrix:ne { #1 } \l_@@_type_of_col_tl
2995   }
2996   { \use:c { end #1 NiceArray } }
2997 }

```

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

```

2998 \cs_new_protected:Npn \@@_NotEmpty:
2999 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

{NiceTabular}, {NiceTabularX} and {NiceTabular*}

```

3000 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3001 {

```

If the dimension \l_@@_width_dim is equal to 0 pt, that means that it has not be set by a previous use of \NiceMatrixOptions.

```

3002   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3003   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3004   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3005   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3006   \bool_set_true:N \l_@@_NiceTabular_bool
3007   \NiceArray { #2 }
3008 }
3009 { \endNiceArray }

3010 \cs_set_protected:Npn \@@_newcolumnntype #1
3011 {
3012   \cs_if_free:cT { NC @ find @ #1 }
3013   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3014   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
3015   \peek_meaning:NTF [
3016     { \newcol@ #1 }
3017     { \newcol@ #1 [ 0 ] }
3018   }

3019 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3020 {

```

The following code prevents the expansion of the ‘X’ columns with the definition of that columns in `tabularx` (this would result in an error in `{NiceTabularX}`).

```

3021 \bool_if:NT \c_@@_tabularx_loaded_bool { \newcolumnntype { X } { \@@_X } }
3022 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3023 \dim_zero_new:N \l_@@_width_dim
3024 \dim_set:Nn \l_@@_width_dim { #1 }
3025 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3026 \bool_set_true:N \l_@@_NiceTabular_bool
3027 \NiceArray { #3 }
3028 }
3029 { \endNiceArray }

3030 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3031 {
3032   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3033   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3034   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3035   \bool_set_true:N \l_@@_NiceTabular_bool
3036   \NiceArray { #3 }
3037 }
3038 { \endNiceArray }

```

After the construction of the array

```

3039 \cs_new_protected:Npn \@@_after_array:
3040 {
3041   \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.

```

3042 \bool_if:NT \g_@@_last_col_found_bool
3043 { \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`.

```

3044 \bool_if:NT \l_@@_last_col_without_value_bool
3045 { \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.

```

3046 \bool_if:NT \l_@@_last_row_without_value_bool
3047 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3048 \tl_gput_right:Nx \g_@@_aux_tl
3049 {
3050   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3051   {
3052     \int_use:N \l_@@_first_row_int ,
3053     \int_use:N \c_iRow ,
3054     \int_use:N \g_@@_row_total_int ,
3055     \int_use:N \l_@@_first_col_int ,
3056     \int_use:N \c_jCol ,
3057     \int_use:N \g_@@_col_total_int
3058   }
3059 }

```

We write also the potential content of `\g_@@_pos_of_blocks_seq`. It will be used to recreate the blocks with a name in the `\CodeBefore` and also if the command `\rowcolors` is used with the key `respect-blocks`).

```

3060 \seq_if_empty:NF \g_@@_pos_of_blocks_seq

```

```

3061 {
3062   \tl_gput_right:Nx \g_@@_aux_tl
3063   {
3064     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3065     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3066   }
3067 }
3068 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3069 {
3070   \tl_gput_right:Nx \g_@@_aux_tl
3071   {
3072     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3073     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3074     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3075     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3076   }
3077 }

```

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

```

3078   \@@_create_diag_nodes:

```

We create the aliases using `last` for the nodes of the cells in the last row and the last column.

```

3079   \pgfpicture
3080   \int_step_inline:nn \c@iRow
3081   {
3082     \pgfnodealias
3083     { \@@_env: - ##1 - last }
3084     { \@@_env: - ##1 - \int_use:N \c@jCol }
3085   }
3086   \int_step_inline:nn \c@jCol
3087   {
3088     \pgfnodealias
3089     { \@@_env: - last - ##1 }
3090     { \@@_env: - \int_use:N \c@iRow - ##1 }
3091   }
3092   \str_if_empty:NF \l_@@_name_str
3093   {
3094     \int_step_inline:nn \c@iRow
3095     {
3096       \pgfnodealias
3097       { \l_@@_name_str - ##1 - last }
3098       { \@@_env: - ##1 - \int_use:N \c@jCol }
3099     }
3100     \int_step_inline:nn \c@jCol
3101     {
3102       \pgfnodealias
3103       { \l_@@_name_str - last - ##1 }
3104       { \@@_env: - \int_use:N \c@iRow - ##1 }
3105     }
3106   }
3107   \endpgfpicture

```

By default, the diagonal lines will be parallelized⁶⁸. 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.

```

3108   \bool_if:NT \l_@@_parallelize_diags_bool
3109   {
3110     \int_gzero_new:N \g_@@_ddots_int
3111     \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`

⁶⁸It's possible to use the option `parallelize-diags` to disable this parallelization.

diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```

3112     \dim_gzero_new:N \g_@@_delta_x_one_dim
3113     \dim_gzero_new:N \g_@@_delta_y_one_dim
3114     \dim_gzero_new:N \g_@@_delta_x_two_dim
3115     \dim_gzero_new:N \g_@@_delta_y_two_dim
3116   }
3117   \int_zero_new:N \l_@@_initial_i_int
3118   \int_zero_new:N \l_@@_initial_j_int
3119   \int_zero_new:N \l_@@_final_i_int
3120   \int_zero_new:N \l_@@_final_j_int
3121   \bool_set_false:N \l_@@_initial_open_bool
3122   \bool_set_false:N \l_@@_final_open_bool

```

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

```

3123   \bool_if:NT \l_@@_small_bool
3124   {
3125     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3126     \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_end_dim` correspond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.

```

3127     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3128     { 0.6 \l_@@_xdots_shorten_start_dim }
3129     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3130     { 0.6 \l_@@_xdots_shorten_end_dim }
3131   }

```

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

```

3132   \@@_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.

```

3133   \@@_compute_corners:

```

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

```

3134   \@@_adjust_pos_of_blocks_seq:
3135   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3136   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3137   \bool_if:NT \c_@@_tikz_loaded_bool
3138   {
3139     \tikzset
3140     {
3141       every~picture / .style =
3142       {
3143         overlay ,
3144         remember~picture ,
3145         name~prefix = \@@_env: -
3146       }
3147     }
3148   }
3149   \cs_set_eq:NN \ialign \@@_old_ialign:
3150   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3151   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3152   \cs_set_eq:NN \OverBrace \@@_OverBrace
3153   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames

```

```

3154 \cs_set_eq:NN \line \@@_line
3155 \g_@@_internal_code_after_tl
3156 \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.

```

3157 \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).

```

3158 \seq_gclear:N \g_@@_submatrix_names_seq

```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```

3159 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3160 \scan_stop:
3161 \tl_gclear:N \g_nicematrix_code_after_tl
3162 \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.

```

3163 \tl_if_empty:NF \g_nicematrix_code_before_tl
3164 {

```

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`.

```

3165 \cs_set_protected:Npn \rectanglecolor { }
3166 \cs_set_protected:Npn \columncolor { }
3167 \tl_gput_right:Nx \g_@@_aux_tl
3168 {
3169 \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3170 { \exp_not:V \g_nicematrix_code_before_tl }
3171 }
3172 \bool_set_true:N \l_@@_code_before_bool
3173 }

```

```

3174 \str_gclear:N \g_@@_name_env_str
3175 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁶⁹. 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`.

```

3176 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3177 }

```

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`.

```

3178 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3179 { { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } } }

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format *i-j*. However, the user is allowed to omit *i* or *j* (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in

⁶⁹e.g. `\color[rgb]{0.5,0.5,0}`

`\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).

```

3180 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3181 {
3182   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3183   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3184 }

```

The following command must *not* be protected.

```

3185 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3186 {
3187   { #1 }
3188   { #2 }
3189   {
3190     \int_compare:nNnTF { #3 } > { 99 }
3191     { \int_use:N \c@iRow }
3192     { #3 }
3193   }
3194   {
3195     \int_compare:nNnTF { #4 } > { 99 }
3196     { \int_use:N \c@jCol }
3197     { #4 }
3198   }
3199   { #5 }
3200 }

```

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).

```

3201 \hook_gput_code:nnn { begindocument } { . }
3202 {
3203   \cs_new_protected:Npx \@@_draw_dotted_lines:
3204   {
3205     \c_@@_pgfortikzpicture_tl
3206     \@@_draw_dotted_lines_i:
3207     \c_@@_endpgfortikzpicture_tl
3208   }
3209 }

```

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

```

3210 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3211 {
3212   \pgfrememberpicturerepositiononpagetrue
3213   \pgf@relevantforpicturesizefalse
3214   \g_@@_HVdotsfor_lines_tl
3215   \g_@@_Vdots_lines_tl
3216   \g_@@_Ddots_lines_tl
3217   \g_@@_Idots_lines_tl
3218   \g_@@_Cdots_lines_tl
3219   \g_@@_Ldots_lines_tl
3220 }

3221 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3222 {
3223   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3224   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3225 }

```

We define a new PGF shape for the diag nodes because we want to provide a anchor called `.5` for those nodes.

```

3226 \pgfdeclareshape { @@_diag_node }

```

```

3227 {
3228   \savedanchor { \five }
3229   {
3230     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3231     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3232   }
3233   \anchor { 5 } { \five }
3234   \anchor { center } { \pgfpointorigin }
3235 }

```

The following command creates the diagonal nodes (in fact, if the matrix is not a square matrix, not all the nodes are on the diagonal).

```

3236 \cs_new_protected:Npn \@@_create_diag_nodes:
3237 {
3238   \pgfpicture
3239   \pgfrememberpicturepositiononpagetrue
3240   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3241   {
3242     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3243     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3244     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3245     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3246     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3247     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3248     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3249     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3250     \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.

```

3251     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3252     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3253     \pgfnode { @_diag_node } { center } { } { \@@_env: - ##1 } { }
3254     \str_if_empty:NF \l_@@_name_str
3255     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3256   }

```

Now, the last node. Of course, that is only a coordinate because there is not `.5` anchor for that node.

```

3257   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3258   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3259   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3260   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3261   \pgfcoordinate
3262   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3263   \pgfnodealias
3264   { \@@_env: - last }
3265   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3266   \str_if_empty:NF \l_@@_name_str
3267   {
3268     \pgfnodealias
3269     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3270     { \@@_env: - \int_use:N \l_tmpa_int }
3271     \pgfnodealias
3272     { \l_@@_name_str - last }
3273     { \@@_env: - last }
3274   }
3275   \endpgfpicture
3276 }

```

We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on

its right.

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

The command `\l_@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
3277 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3278 {
```

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

```
3279 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3280 \int_set:Nn \l_@@_initial_i_int { #1 }
3281 \int_set:Nn \l_@@_initial_j_int { #2 }
3282 \int_set:Nn \l_@@_final_i_int { #1 }
3283 \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.

```
3284 \bool_set_false:N \l_@@_stop_loop_bool
3285 \bool_do_until:Nn \l_@@_stop_loop_bool
3286 {
3287   \int_add:Nn \l_@@_final_i_int { #3 }
3288   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3289 \bool_set_false:N \l_@@_final_open_bool
3290 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3291 {
3292   \int_compare:nNnTF { #3 } = 1
3293   { \bool_set_true:N \l_@@_final_open_bool }
3294   {
3295     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3296     { \bool_set_true:N \l_@@_final_open_bool }
3297   }
3298 }
3299 {
3300   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3301   {
3302     \int_compare:nNnTF { #4 } = { -1 }
3303     { \bool_set_true:N \l_@@_final_open_bool }
3304   }
3305 }
```

```

3306         \int_compare:nNtT \l_@@_final_j_int > \l_@@_col_max_int
3307         {
3308             \int_compare:nNtT { #4 } = 1
3309             { \bool_set_true:N \l_@@_final_open_bool }
3310         }
3311     }
3312 }
3313 \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.

```

3314 {

```

We do a step backwards.

```

3315     \int_sub:Nn \l_@@_final_i_int { #3 }
3316     \int_sub:Nn \l_@@_final_j_int { #4 }
3317     \bool_set_true:N \l_@@_stop_loop_bool
3318 }

```

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`.

```

3319 {
3320     \cs_if_exist:cTF
3321     {
3322         @@ _ dotted _
3323         \int_use:N \l_@@_final_i_int -
3324         \int_use:N \l_@@_final_j_int
3325     }
3326     {
3327         \int_sub:Nn \l_@@_final_i_int { #3 }
3328         \int_sub:Nn \l_@@_final_j_int { #4 }
3329         \bool_set_true:N \l_@@_final_open_bool
3330         \bool_set_true:N \l_@@_stop_loop_bool
3331     }
3332     {
3333         \cs_if_exist:cTF
3334         {
3335             pgf @ sh @ ns @ \@@_env:
3336             - \int_use:N \l_@@_final_i_int
3337             - \int_use:N \l_@@_final_j_int
3338         }
3339         { \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.

```

3340     {
3341         \cs_set:cpn
3342         {
3343             @@ _ dotted _
3344             \int_use:N \l_@@_final_i_int -
3345             \int_use:N \l_@@_final_j_int
3346         }
3347         { }
3348     }
3349 }
3350 }
3351 }

```

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

```

3352     \bool_set_false:N \l_@@_stop_loop_bool

```

```

3353 \bool_do_until:Nn \l_@@_stop_loop_bool
3354 {
3355   \int_sub:Nn \l_@@_initial_i_int { #3 }
3356   \int_sub:Nn \l_@@_initial_j_int { #4 }
3357   \bool_set_false:N \l_@@_initial_open_bool
3358   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3359   {
3360     \int_compare:nNnTF { #3 } = 1
3361     { \bool_set_true:N \l_@@_initial_open_bool }
3362     {
3363       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3364       { \bool_set_true:N \l_@@_initial_open_bool }
3365     }
3366   }
3367   {
3368     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3369     {
3370       \int_compare:nNnT { #4 } = 1
3371       { \bool_set_true:N \l_@@_initial_open_bool }
3372     }
3373     {
3374       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3375       {
3376         \int_compare:nNnT { #4 } = { -1 }
3377         { \bool_set_true:N \l_@@_initial_open_bool }
3378       }
3379     }
3380   }
3381   \bool_if:NnTF \l_@@_initial_open_bool
3382   {
3383     \int_add:Nn \l_@@_initial_i_int { #3 }
3384     \int_add:Nn \l_@@_initial_j_int { #4 }
3385     \bool_set_true:N \l_@@_stop_loop_bool
3386   }
3387   {
3388     \cs_if_exist:cTF
3389     {
3390       @@ _ dotted _
3391       \int_use:N \l_@@_initial_i_int -
3392       \int_use:N \l_@@_initial_j_int
3393     }
3394     {
3395       \int_add:Nn \l_@@_initial_i_int { #3 }
3396       \int_add:Nn \l_@@_initial_j_int { #4 }
3397       \bool_set_true:N \l_@@_initial_open_bool
3398       \bool_set_true:N \l_@@_stop_loop_bool
3399     }
3400   }
3401   \cs_if_exist:cTF
3402   {
3403     pgf @ sh @ ns @ \@@_env:
3404     - \int_use:N \l_@@_initial_i_int
3405     - \int_use:N \l_@@_initial_j_int
3406   }
3407   { \bool_set_true:N \l_@@_stop_loop_bool }
3408   {
3409     \cs_set:cpn
3410     {
3411       @@ _ dotted _
3412       \int_use:N \l_@@_initial_i_int -
3413       \int_use:N \l_@@_initial_j_int
3414     }
3415     { }

```

```

3416         }
3417     }
3418 }
3419 }

```

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.

```

3420 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3421 {
3422     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

3423     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3424     { \int_use:N \l_@@_final_i_int }
3425     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3426     { } % for the name of the block
3427 }
3428 }

```

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).

```

3429 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3430 {
3431     \int_set:Nn \l_@@_row_min_int 1
3432     \int_set:Nn \l_@@_col_min_int 1
3433     \int_set_eq:NN \l_@@_row_max_int \c{iRow}
3434     \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`.

```

3435 \seq_map_inline:Nn \g_@@_submatrix_seq
3436 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
3437 }

```

#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.

```

3438 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3439 {
3440     \bool_if:nT
3441     {
3442         \int_compare_p:n { #3 <= #1 }
3443         && \int_compare_p:n { #1 <= #5 }
3444         && \int_compare_p:n { #4 <= #2 }
3445         && \int_compare_p:n { #2 <= #6 }
3446     }
3447     {
3448         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3449         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3450         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3451         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3452     }
3453 }

```

```

3454 \cs_new_protected:Npn \@@_set_initial_coords:
3455 {
3456     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3457     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3458 }
3459 \cs_new_protected:Npn \@@_set_final_coords:
3460 {

```

```

3461 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3462 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3463 }
3464 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3465 {
3466   \pgfpointanchor
3467   {
3468     \@@_env:
3469     - \int_use:N \l_@@_initial_i_int
3470     - \int_use:N \l_@@_initial_j_int
3471   }
3472   { #1 }
3473   \@@_set_initial_coords:
3474 }
3475 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3476 {
3477   \pgfpointanchor
3478   {
3479     \@@_env:
3480     - \int_use:N \l_@@_final_i_int
3481     - \int_use:N \l_@@_final_j_int
3482   }
3483   { #1 }
3484   \@@_set_final_coords:
3485 }
3486 \cs_new_protected:Npn \@@_open_x_initial_dim:
3487 {
3488   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3489   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3490   {
3491     \cs_if_exist:cT
3492     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3493     {
3494       \pgfpointanchor
3495       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3496       { west }
3497       \dim_set:Nn \l_@@_x_initial_dim
3498       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3499     }
3500   }
3501   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3502   {
3503     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3504     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3505     \dim_add:Nn \l_@@_x_initial_dim \col@sep
3506   }
3507 }
3508 \cs_new_protected:Npn \@@_open_x_final_dim:
3509 {
3510   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3511   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3512   {
3513     \cs_if_exist:cT
3514     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3515     {
3516       \pgfpointanchor
3517       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3518       { east }
3519       \dim_set:Nn \l_@@_x_final_dim
3520       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3521     }

```

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

```
3522 }
```

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

```
3523 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3524 {
3525   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
3526   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3527   \dim_sub:Nn \l_@@_x_final_dim \col@sep
3528 }
3529 }
```

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.

```
3530 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3531 {
3532   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3533   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3534   {
3535     \@@_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.

```
3536   \group_begin:
3537   \int_compare:nNnTF { #1 } = 0
3538     { \color { nicematrix-first-row } }
3539     {
```

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.

```
3540     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3541     { \color { nicematrix-last-row } }
3542   }
3543   \keys_set:nn { NiceMatrix / xdots } { #3 }
3544   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3545   \@@_actually_draw_Ldots:
3546   \group_end:
3547 }
3548 }
```

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`.

```
3549 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3550 {
3551   \bool_if:NTF \l_@@_initial_open_bool
3552   {
3553     \@@_open_x_initial_dim:
3554     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3555     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3556   }
3557   { \@@_set_initial_coords_from_anchor:n { base-east } }
3558   \bool_if:NTF \l_@@_final_open_bool
```



```

3559 {
3560   \@@_open_x_final_dim:
3561   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3562   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3563 }
3564 { \@@_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 (?).

```

3565   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
3566   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
3567   \@@_draw_line:
3568 }

```

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.

```

3569 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3570 {
3571   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3572   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3573   {
3574     \@@_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.

```

3575     \group_begin:
3576     \int_compare:nNnTF { #1 } = 0
3577     { \color { nicematrix-first-row } }
3578     {

```

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.

```

3579         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3580         { \color { nicematrix-last-row } }
3581     }
3582     \keys_set:nn { NiceMatrix / xdots } { #3 }
3583     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3584     \@@_actually_draw_Cdots:
3585   \group_end:
3586 }
3587 }

```

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`.

```

3588 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3589 {
3590   \bool_if:NTF \l_@@_initial_open_bool
3591   { \@@_open_x_initial_dim: }
3592   { \@@_set_initial_coords_from_anchor:n { mid~east } }
3593   \bool_if:NTF \l_@@_final_open_bool
3594   { \@@_open_x_final_dim: }
3595   { \@@_set_final_coords_from_anchor:n { mid~west } }
3596   \bool_lazy_and:nnTF

```

```

3597 \l_@@_initial_open_bool
3598 \l_@@_final_open_bool
3599 {
3600   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3601   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3602   \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
3603   \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3604   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3605 }
3606 {
3607   \bool_if:NT \l_@@_initial_open_bool
3608   { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3609   \bool_if:NT \l_@@_final_open_bool
3610   { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3611 }
3612 \@@_draw_line:
3613 }
3614 \cs_new_protected:Npn \@@_open_y_initial_dim:
3615 {
3616   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3617   \dim_set:Nn \l_@@_y_initial_dim
3618   { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3619   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3620   {
3621     \cs_if_exist:cT
3622     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3623     {
3624       \pgfpointanchor
3625       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3626       { north }
3627       \dim_set:Nn \l_@@_y_initial_dim
3628       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3629     }
3630   }
3631 }
3632 \cs_new_protected:Npn \@@_open_y_final_dim:
3633 {
3634   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3635   \dim_set:Nn \l_@@_y_final_dim
3636   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3637   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3638   {
3639     \cs_if_exist:cT
3640     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3641     {
3642       \pgfpointanchor
3643       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3644       { south }
3645       \dim_set:Nn \l_@@_y_final_dim
3646       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3647     }
3648   }
3649 }

```

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.

```

3650 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3651 {
3652   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3653   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3654   {
3655     \@@_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.

```

3656     \group_begin:
3657     \int_compare:nNnTF { #2 } = 0
3658     { \color { nicematrix-first-col } }
3659     {
3660     \int_compare:nNnT { #2 } = \l_@@_last_col_int
3661     { \color { nicematrix-last-col } }
3662     }
3663     \keys_set:nn { NiceMatrix / xdots } { #3 }
3664     \tl_if_empty:VF \l_@@_xdots_color_tl
3665     { \color { \l_@@_xdots_color_tl } }
3666     \@@_actually_draw_Vdots:
3667 \group_end:
3668 }
3669 }

```

The command `\@@_actually_draw_Vdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

The following function is also used by `\Vdotsfor`.

```

3670 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3671 {

```

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

```

3672     \bool_set_false:N \l_tmpa_bool

```

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

```

3673     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3674     {
3675     \@@_set_initial_coords_from_anchor:n { south-west }
3676     \@@_set_final_coords_from_anchor:n { north-west }
3677     \bool_set:Nn \l_tmpa_bool
3678     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3679     }

```

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

```

3680     \bool_if:NTF \l_@@_initial_open_bool
3681     \@@_open_y_initial_dim:
3682     { \@@_set_initial_coords_from_anchor:n { south } }
3683     \bool_if:NTF \l_@@_final_open_bool
3684     \@@_open_y_final_dim:
3685     { \@@_set_final_coords_from_anchor:n { north } }
3686     \bool_if:NTF \l_@@_initial_open_bool
3687     {
3688     \bool_if:NTF \l_@@_final_open_bool
3689     {
3690     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3691     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3692     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
3693     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3694     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim

```

We may think that the final user won't use a "last column" which contains only a command `\Vdots`. However, if the `\Vdots` is in fact used to draw, not a dotted line, but an arrow (to indicate the number of rows of the matrix), it may be really encountered.

```

3695         \int_compare:nNnT \l_@@_last_col_int > { -2 }
3696         {
3697             \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3698             {
3699                 \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3700                 \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3701                 \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3702                 \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3703             }
3704         }
3705     }
3706     { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3707 }
3708 {
3709     \bool_if:NTF \l_@@_final_open_bool
3710     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3711     {

```

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.

```

3712         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3713         {
3714             \dim_set:Nn \l_@@_x_initial_dim
3715             {
3716                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3717                 \l_@@_x_initial_dim \l_@@_x_final_dim
3718             }
3719             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3720         }
3721     }
3722 }
3723 \@@_draw_line:
3724 }

```

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.

```

3725 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3726 {
3727     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3728     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3729     {
3730         \@@_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.

```

3731     \group_begin:
3732     \keys_set:nn { NiceMatrix / xdots } { #3 }
3733     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3734     \@@_actually_draw_Ddots:
3735     \group_end:
3736 }
3737 }

```

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.`

```

3738 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3739 {
3740   \bool_if:NTF \l_@@_initial_open_bool
3741   {
3742     \@@_open_y_initial_dim:
3743     \@@_open_x_initial_dim:
3744   }
3745   { \@@_set_initial_coords_from_anchor:n { south-east } }
3746   \bool_if:NTF \l_@@_final_open_bool
3747   {
3748     \@@_open_x_final_dim:
3749     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3750   }
3751   { \@@_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.

```

3752   \bool_if:NT \l_@@_parallelize_diags_bool
3753   {
3754     \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).

```

3755     \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.

```

3756     {
3757       \dim_gset:Nn \g_@@_delta_x_one_dim
3758       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3759       \dim_gset:Nn \g_@@_delta_y_one_dim
3760       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3761     }

```

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`.

```

3762     {
3763       \dim_set:Nn \l_@@_y_final_dim
3764       {
3765         \l_@@_y_initial_dim +
3766         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3767         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3768       }
3769     }
3770   }
3771   \@@_draw_line:
3772 }

```

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.

```

3773 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3774 {
3775   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3776   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }

```

```

3777 {
3778   \@@_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.

```

3779   \group_begin:
3780     \keys_set:nn { NiceMatrix / xdots } { #3 }
3781     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3782     \@@_actually_draw_Iddots:
3783   \group_end:
3784 }
3785 }

```

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.`

```

3786 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3787 {
3788   \bool_if:NTF \l_@@_initial_open_bool
3789   {
3790     \@@_open_y_initial_dim:
3791     \@@_open_x_initial_dim:
3792   }
3793   { \@@_set_initial_coords_from_anchor:n { south-west } }
3794   \bool_if:NTF \l_@@_final_open_bool
3795   {
3796     \@@_open_y_final_dim:
3797     \@@_open_x_final_dim:
3798   }
3799   { \@@_set_final_coords_from_anchor:n { north-east } }
3800   \bool_if:NT \l_@@_parallelize_diags_bool
3801   {
3802     \int_gincr:N \g_@@_iddots_int
3803     \int_compare:nNnTF \g_@@_iddots_int = 1
3804     {
3805       \dim_gset:Nn \g_@@_delta_x_two_dim
3806       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3807       \dim_gset:Nn \g_@@_delta_y_two_dim
3808       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3809     }
3810     {
3811       \dim_set:Nn \l_@@_y_final_dim
3812       {
3813         \l_@@_y_initial_dim +
3814         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3815         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3816       }
3817     }
3818   }
3819   \@@_draw_line:
3820 }

```

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`

```

3821 \cs_new_protected:Npn \@@_draw_line:
3822 {
3823   \pgfrememberpicturepositiononpagetrue
3824   \pgf@relevantforpicturesizefalse
3825   \bool_lazy_or:nnTF
3826   { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }

```

The boolean `\l_@@_dotted_bool` is raised for the rules specified by either `\hdottedline` or `:` (or the letter specified by `letter-for-dotted-lines`) in the preamble of the array.

```

3827   \l_@@_dotted_bool
3828   \@@_draw_standard_dotted_line:
3829   \@@_draw_unstandard_dotted_line:
3830 }

```

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.

```

3831 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
3832 {
3833   \begin { scope }
3834   \@@_draw_unstandard_dotted_line:o
3835   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3836 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

3837 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
3838 {
3839   \@@_draw_unstandard_dotted_line:nVV
3840   { #1 }
3841   \l_@@_xdots_up_tl
3842   \l_@@_xdots_down_tl
3843 }
3844 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
3845 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
3846 {
3847   \draw
3848   [
3849     #1 ,
3850     shorten-> = \l_@@_xdots_shorten_end_dim ,
3851     shorten-< = \l_@@_xdots_shorten_start_dim ,
3852   ]
3853   ( \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).

```

3854   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3855   node [ sloped , below ] { $ \scriptstyle #3 $ }
3856   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;

```

```

3857 \end { scope }
3858 }
3859 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnn { n V V }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real round dots).

```

3860 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3861 {
3862   \bool_lazy_and:nnF
3863   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3864   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3865   {
3866     \pgfscope
3867     \pgftransformshift
3868     {
3869       \pgfpointlineattime { 0.5 }
3870       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3871       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3872     }
3873     \pgftransformrotate
3874     {
3875       \fp_eval:n
3876       {
3877         atand
3878         (
3879           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3880           \l_@@_x_final_dim - \l_@@_x_initial_dim
3881         )
3882       }
3883     }
3884     \pgfnode
3885     { rectangle }
3886     { south }
3887     {
3888       \c_math_toggle_token
3889       \scriptstyle \l_@@_xdots_up_tl
3890       \c_math_toggle_token
3891     }
3892     { }
3893     { \pgfusepath { } }
3894     \pgfnode
3895     { rectangle }
3896     { north }
3897     {
3898       \c_math_toggle_token
3899       \scriptstyle \l_@@_xdots_down_tl
3900       \c_math_toggle_token
3901     }
3902     { }
3903     { \pgfusepath { } }
3904     \endpgfscope
3905   }
3906   \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.

```

3907   \dim_zero_new:N \l_@@_l_dim
3908   \dim_set:Nn \l_@@_l_dim
3909   {
3910     \fp_to_dim:n
3911     {
3912       sqrt
3913       (

```



```

3914          ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3915          +
3916          ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3917      )
3918  }
3919  }

```

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.

```

3920      \bool_lazy_or:nnF
3921      { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3922      { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3923      \@@_draw_standard_dotted_line_i:
3924  \group_end:
3925  }
3926  \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3927  \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3928  {

```

The number of dots will be `\l_tmpa_int + 1`.

```

3929      \bool_if:NTF \l_@@_initial_open_bool
3930      {
3931          \bool_if:NTF \l_@@_final_open_bool
3932          {
3933              \int_set:Nn \l_tmpa_int
3934              { \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim }
3935          }
3936          {
3937              \int_set:Nn \l_tmpa_int
3938              {
3939                  \dim_ratio:nn
3940                  { \l_@@_l_dim - \l_@@_xdots_shorten_start_dim }
3941                  \l_@@_xdots_inter_dim
3942              }
3943          }
3944      }
3945      {
3946          \bool_if:NTF \l_@@_final_open_bool
3947          {
3948              \int_set:Nn \l_tmpa_int
3949              {
3950                  \dim_ratio:nn
3951                  { \l_@@_l_dim - \l_@@_xdots_shorten_end_dim }
3952                  \l_@@_xdots_inter_dim
3953              }
3954          }
3955          {
3956              \int_set:Nn \l_tmpa_int
3957              {
3958                  \dim_ratio:nn
3959                  {
3960                      \l_@@_l_dim
3961                      - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
3962                  }
3963                  \l_@@_xdots_inter_dim
3964              }
3965          }
3966      }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

3967 \dim_set:Nn \l_tmpa_dim
3968 {
3969   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3970   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
3971 }
3972 \dim_set:Nn \l_tmpb_dim
3973 {
3974   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3975   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
3976 }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

3977 \dim_gadd:Nn \l_@@_x_initial_dim
3978 {
3979   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3980   \dim_ratio:nn
3981   {
3982     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
3983     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
3984   }
3985   { 2 \l_@@_l_dim }
3986 }
3987 \dim_gadd:Nn \l_@@_y_initial_dim
3988 {
3989   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3990   \dim_ratio:nn
3991   {
3992     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
3993     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
3994   }
3995   { 2 \l_@@_l_dim }
3996 }
3997 \pgf@relevantforpicturesizefalse
3998 \int_step_inline:nnn 0 \l_tmpa_int
3999 {
4000   \pgfpathcircle
4001   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4002   { \l_@@_xdots_radius_dim }
4003   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4004   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4005 }
4006 \pgfusepathqfill
4007 }

```

User commands available in the new environments

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

4008 \hook_gput_code:nnn { begindocument } { . }
4009 {
4010   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }

```

```

4011 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4012 \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
4013 {
4014   \int_compare:nNnTF \c@jCol = 0
4015   { \@@_error:nn { in~first~col } \Ldots }
4016   {
4017     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4018     { \@@_error:nn { in~last~col } \Ldots }
4019     {
4020       \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
4021       { #1 , down = #2 , up = #3 }
4022     }
4023   }
4024   \bool_if:NF \l_@@_nullify_dots_bool
4025   { \phantom { \ensuremath { \@@_old_ldots } } }
4026   \bool_gset_true:N \g_@@_empty_cell_bool
4027 }

4028 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4029 {
4030   \int_compare:nNnTF \c@jCol = 0
4031   { \@@_error:nn { in~first~col } \Cdots }
4032   {
4033     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4034     { \@@_error:nn { in~last~col } \Cdots }
4035     {
4036       \@@_instruction_of_type:nnn \c_false_bool { \Cdots }
4037       { #1 , down = #2 , up = #3 }
4038     }
4039   }
4040   \bool_if:NF \l_@@_nullify_dots_bool
4041   { \phantom { \ensuremath { \@@_old_cdots } } }
4042   \bool_gset_true:N \g_@@_empty_cell_bool
4043 }

4044 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4045 {
4046   \int_compare:nNnTF \c@iRow = 0
4047   { \@@_error:nn { in~first~row } \Vdots }
4048   {
4049     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4050     { \@@_error:nn { in~last~row } \Vdots }
4051     {
4052       \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4053       { #1 , down = #2 , up = #3 }
4054     }
4055   }
4056   \bool_if:NF \l_@@_nullify_dots_bool
4057   { \phantom { \ensuremath { \@@_old_vdots } } }
4058   \bool_gset_true:N \g_@@_empty_cell_bool
4059 }

4060 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4061 {
4062   \int_case:nnF \c@iRow
4063   {
4064     0 { \@@_error:nn { in~first~row } \Ddots }
4065     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4066   }
4067   {
4068     \int_case:nnF \c@jCol

```

```

4069         {
4070             0 { \@@_error:nn { in~first~col } \Ddots }
4071             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4072         }
4073         {
4074             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4075             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4076             { #1 , down = #2 , up = #3 }
4077         }
4078     }
4079 }
4080 \bool_if:NF \l_@@_nullify_dots_bool
4081 { \phantom { \ensuremath { \@@_old_ddots } } }
4082 \bool_gset_true:N \g_@@_empty_cell_bool
4083 }

4084 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4085 {
4086     \int_case:nnF \c@iRow
4087     {
4088         0 { \@@_error:nn { in~first~row } \Iddots }
4089         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4090     }
4091     {
4092         \int_case:nnF \c@jCol
4093         {
4094             0 { \@@_error:nn { in~first~col } \Iddots }
4095             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4096         }
4097         {
4098             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4099             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4100             { #1 , down = #2 , up = #3 }
4101         }
4102     }
4103     \bool_if:NF \l_@@_nullify_dots_bool
4104     { \phantom { \ensuremath { \@@_old_iddots } } }
4105     \bool_gset_true:N \g_@@_empty_cell_bool
4106 }
4107 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

4108 \keys_define:nn { NiceMatrix / Ddots }
4109 {
4110     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4111     draw-first .default:n = true ,
4112     draw-first .value_forbidden:n = true
4113 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

4114 \cs_new_protected:Npn \@@_Hspace:
4115 {
4116     \bool_gset_true:N \g_@@_empty_cell_bool
4117     \hspace
4118 }

```

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.

```

4119 \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`.

```

4120 \cs_new:Npn \@@Hdotsfor:
4121 {
4122   \bool_lazy_and:nnTF
4123     { \int_compare_p:nNn \c@jCol = 0 }
4124     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4125     {
4126       \bool_if:NTF \g_@@_after_col_zero_bool
4127       {
4128         \multicolumn { 1 } { c } { }
4129         \@@Hdotsfor_i
4130       }
4131       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4132     }
4133     {
4134       \multicolumn { 1 } { c } { }
4135       \@@Hdotsfor_i
4136     }
4137 }

```

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:`).

```

4138 \hook_gput_code:nnn { begindocument } { . }
4139 {
4140   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4141   \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.

```

4142   \exp_args:NNV \NewDocumentCommand \@@Hdotsfor_i \l_@@_argspec_tl
4143   {
4144     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4145     {
4146       \@@Hdotsfor:nnnn
4147       { \int_use:N \c@iRow }
4148       { \int_use:N \c@jCol }
4149       { #2 }
4150       {
4151         #1 , #3 ,
4152         down = \exp_not:n { #4 } ,
4153         up = \exp_not:n { #5 }
4154       }
4155     }
4156     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4157   }
4158 }

```

Enf of `\AddToHook`.

```

4159 \cs_new_protected:Npn \@@Hdotsfor:nnnn #1 #2 #3 #4
4160 {
4161   \bool_set_false:N \l_@@_initial_open_bool
4162   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4163   \int_set:Nn \l_@@_initial_i_int { #1 }
4164   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

4165   \int_compare:nNnTF { #2 } = 1
4166   {

```

```

4167     \int_set:Nn \l_@@_initial_j_int 1
4168     \bool_set_true:N \l_@@_initial_open_bool
4169   }
4170   {
4171     \cs_if_exist:cTF
4172     {
4173       pgf @ sh @ ns @ \@@_env:
4174       - \int_use:N \l_@@_initial_i_int
4175       - \int_eval:n { #2 - 1 }
4176     }
4177     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4178     {
4179       \int_set:Nn \l_@@_initial_j_int { #2 }
4180       \bool_set_true:N \l_@@_initial_open_bool
4181     }
4182   }
4183   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4184   {
4185     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4186     \bool_set_true:N \l_@@_final_open_bool
4187   }
4188   {
4189     \cs_if_exist:cTF
4190     {
4191       pgf @ sh @ ns @ \@@_env:
4192       - \int_use:N \l_@@_final_i_int
4193       - \int_eval:n { #2 + #3 }
4194     }
4195     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4196     {
4197       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4198       \bool_set_true:N \l_@@_final_open_bool
4199     }
4200   }
4201   \group_begin:
4202   \int_compare:nNnTF { #1 } = 0
4203   { \color { nicematrix-first-row } }
4204   {
4205     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4206     { \color { nicematrix-last-row } }
4207   }
4208   \keys_set:nn { NiceMatrix / xdots } { #4 }
4209   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4210   \@@_actually_draw_Ldots:
4211   \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).

```

4212     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4213     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4214   }

4215   \hook_gput_code:nnn { begindocument } { . }
4216   {
4217     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4218     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4219     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4220     {
4221       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4222       {
4223         \@@_Vdotsfor:nnnn

```

```

4224         { \int_use:N \c@iRow }
4225         { \int_use:N \c@jCol }
4226         { #2 }
4227         {
4228             #1 , #3 ,
4229             down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4230         }
4231     }
4232 }
4233 }

```

Enf of \AddToHook.

```

4234 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4235 {
4236     \bool_set_false:N \l_@@_initial_open_bool
4237     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

4238     \int_set:Nn \l_@@_initial_j_int { #2 }
4239     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

4240     \int_compare:nNnTF #1 = 1
4241     {
4242         \int_set:Nn \l_@@_initial_i_int 1
4243         \bool_set_true:N \l_@@_initial_open_bool
4244     }
4245     {
4246         \cs_if_exist:cTF
4247         {
4248             pgf @ sh @ ns @ \@@_env:
4249             - \int_eval:n { #1 - 1 }
4250             - \int_use:N \l_@@_initial_j_int
4251         }
4252         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4253         {
4254             \int_set:Nn \l_@@_initial_i_int { #1 }
4255             \bool_set_true:N \l_@@_initial_open_bool
4256         }
4257     }
4258     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4259     {
4260         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4261         \bool_set_true:N \l_@@_final_open_bool
4262     }
4263     {
4264         \cs_if_exist:cTF
4265         {
4266             pgf @ sh @ ns @ \@@_env:
4267             - \int_eval:n { #1 + #3 }
4268             - \int_use:N \l_@@_final_j_int
4269         }
4270         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4271         {
4272             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4273             \bool_set_true:N \l_@@_final_open_bool
4274         }
4275     }
4276     \group_begin:
4277     \int_compare:nNnTF { #2 } = 0
4278     { \color { nicematrix-first-col } }
4279     {
4280         \int_compare:nNnT { #2 } = \g_@@_col_total_int
4281         { \color { nicematrix-last-col } }

```

```

4282     }
4283     \keys_set:nn { NiceMatrix / xdots } { #4 }
4284     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4285     \@@_actually_draw_Vdots:
4286     \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).

```

4287     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4288     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4289 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

4290 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

```

The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells.

First, we write a command with the following behaviour:

- If the argument is of the format i - j , our command applies the command `\int_eval:n` to i and j ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).⁷⁰

```

4291 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4292 {
4293     \tl_if_empty:nTF { #2 }
4294     { #1 }
4295     { \@@_double_int_eval_i:n #1-#2 \q_stop }
4296 }
4297 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
4298 { \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).

```

4299 \hook_gput_code:nnn { begindocument } { . }
4300 {
4301     \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4302     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4303     \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4304     {
4305         \group_begin:
4306         \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4307         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4308         \use:e
4309         {
4310             \@@_line_i:nn
4311             { \@@_double_int_eval:n #2 - \q_stop }

```

⁷⁰Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.


```

4312         { \@@_double_int_eval:n #3 - \q_stop }
4313     }
4314     \group_end:
4315 }
4316 }
4317 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4318 {
4319     \bool_set_false:N \l_@@_initial_open_bool
4320     \bool_set_false:N \l_@@_final_open_bool
4321     \bool_if:nTF
4322     {
4323         \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4324         ||
4325         \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4326     }
4327     {
4328         \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4329     }
4330     { \@@_draw_line_ii:nn { #1 } { #2 } }
4331 }
4332 \hook_gput_code:nnn { begindocument } { . }
4333 {
4334     \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4335     {

```

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:`.

```

4336     \c_@@_pgfortikzpicture_tl
4337     \@@_draw_line_iii:nn { #1 } { #2 }
4338     \c_@@_endpgfortikzpicture_tl
4339 }
4340 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

4341 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4342 {
4343     \pgfrememberpicturepositiononpagetrue
4344     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4345     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4346     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4347     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4348     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4349     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4350     \@@_draw_line:
4351 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

The command `\RowStyle`

```

4352 \keys_define:nn { NiceMatrix / RowStyle }
4353 {
4354     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4355     cell-space-top-limit .initial:n = \c_zero_dim ,
4356     cell-space-top-limit .value_required:n = true ,
4357     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4358     cell-space-bottom-limit .initial:n = \c_zero_dim ,
4359     cell-space-bottom-limit .value_required:n = true ,

```

```

4360 cell-space-limits .meta:n =
4361 {
4362     cell-space-top-limit = #1 ,
4363     cell-space-bottom-limit = #1 ,
4364 } ,
4365 color .tl_set:N = \l_tmpa_tl ,
4366 color .value_required:n = true ,
4367 bold .bool_set:N = \l_tmpa_bool ,
4368 bold .default:n = true ,
4369 bold .initial:n = false ,
4370 nb-rows .code:n =
4371     \str_if_eq:nnTF { #1 } { * }
4372     { \int_set_eq:NN \l_@@_key_nb_rows_int 500 }
4373     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
4374 nb-rows .value_required:n = true ,
4375 rowcolor .tl_set:N = \l_@@_tmpc_tl ,
4376 rowcolor .value_required:n = true ,
4377 rowcolor .initial:n = ,
4378 unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
4379 }

```

```

4380 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4381 {
4382     \tl_clear:N \l_tmpa_tl
4383     \int_set:Nn \l_@@_key_nb_rows_int 1
4384     \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

4385     \tl_if_empty:NF \l_@@_tmpc_tl
4386     {

```

First, the end of the current row (we remind that \RowStyle applies to the *end* of the current row).

```

4387         \tl_gput_right:Nx \g_nicematrix_code_before_tl
4388         {
4389             \@@_rectanglecolor
4390             { \l_@@_tmpc_tl }
4391             { \int_use:N \c@iRow - \int_use:N \c@jCol }
4392             { \int_use:N \c@iRow - * }
4393         }

```

Then, the other rows (if there is several rows).

```

4394         \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4395         {
4396             \tl_gput_right:Nx \g_nicematrix_code_before_tl
4397             {
4398                 \@@_rowcolor
4399                 { \l_@@_tmpc_tl }
4400                 {
4401                     \int_eval:n { \c@iRow + 1 }
4402                     - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4403                 }
4404             }
4405         }
4406     }
4407     \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4408     \tl_gput_right:Nx \g_@@_row_style_tl
4409     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4410     \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.

```

4411     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4412     {
4413         \tl_gput_right:Nx \g_@@_row_style_tl
4414         {
4415             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4416             {

```

```

4417         \dim_set:Nn \l_@@_cell_space_top_limit_dim
4418         { \dim_use:N \l_tmpa_dim }
4419     }
4420 }
4421 }
\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.
4422 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4423 {
4424     \tl_gput_right:Nx \g_@@_row_style_tl
4425     {
4426         \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4427         {
4428             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4429             { \dim_use:N \l_tmpb_dim }
4430         }
4431     }
4432 }
\l_tmpa_tl is the value of the key color of \RowStyle.
4433 \tl_if_empty:NF \l_tmpa_tl
4434 {
4435     \tl_gput_right:Nx \g_@@_row_style_tl
4436     { \mode_leave_vertical: \exp_not:N \color { \l_tmpa_tl } }
4437 }
\l_tmpa_bool is the value of the key bold.
4438 \bool_if:NT \l_tmpa_bool
4439 {
4440     \tl_gput_right:Nn \g_@@_row_style_tl
4441     {
4442         \if_mode_math:
4443             \c_math_toggle_token
4444             \bfseries \boldmath
4445             \c_math_toggle_token
4446         \else:
4447             \bfseries \boldmath
4448         \fi:
4449     }
4450 }
4451 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4452 \g_@@_row_style_tl
4453 \ignorespaces
4454 }

```

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).

```
4455 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4456 {
```

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.

```
4457   \int_zero:N \l_tmpa_int
4458   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4459   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4460   \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

First, the case where the color is a *new* color (not in the sequence).

```
4461   {
4462     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4463     \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4464   }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
4465     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4466   }

4467 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4468 \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`).

```
4469 \cs_new_protected:Npn \@@_actually_color:
4470 {
4471   \pgfpicture
4472   \pgf@relevantforpicturesizefalse
4473   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4474   {
4475     \color ##2
4476     \use:c { g_@@_color _ ##1 _tl }
4477     \tl_gclear:c { g_@@_color _ ##1 _tl }
4478     \pgfusepath { fill }
4479   }
4480   \endpgfpicture
4481 }

4482 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4483 {
4484   \tl_set:Nn \l_@@_rows_tl { #1 }
4485   \tl_set:Nn \l_@@_cols_tl { #2 }
4486   \@@_cartesian_path:
4487 }
```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
4488 \NewDocumentCommand \@@_rowcolor { O { } m m }
4489 {
4490   \tl_if_blank:nF { #2 }
4491   {
4492     \@@_add_to_colors_seq:xn
4493     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4494     { \@@_cartesian_color:nn { #3 } { - } }
4495   }
4496 }
```

Here an example : \@@_columncolor:nn {red!15} {1,3,5-7,10-}

```

4497 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4498 {
4499   \tl_if_blank:nF { #2 }
4500   {
4501     \@@_add_to_colors_seq:xn
4502     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4503     { \@@_cartesian_color:nn { - } { #3 } }
4504   }
4505 }

```

Here is an example : \@@_rectanglecolor{red!15}{2-3}{5-6}

```

4506 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4507 {
4508   \tl_if_blank:nF { #2 }
4509   {
4510     \@@_add_to_colors_seq:xn
4511     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4512     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4513   }
4514 }

```

The last argument is the radius of the corners of the rectangle.

```

4515 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4516 {
4517   \tl_if_blank:nF { #2 }
4518   {
4519     \@@_add_to_colors_seq:xn
4520     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4521     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4522   }
4523 }

```

The last argument is the radius of the corners of the rectangle.

```

4524 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4525 {
4526   \@@_cut_on_hyphen:w #1 \q_stop
4527   \tl_clear_new:N \l_@@_tmpc_tl
4528   \tl_clear_new:N \l_@@_tmpd_tl
4529   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
4530   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
4531   \@@_cut_on_hyphen:w #2 \q_stop
4532   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
4533   \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.

```

4534   \@@_cartesian_path:n { #3 }
4535 }

```

Here is an example : \@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}

```

4536 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4537 {
4538   \clist_map_inline:nn { #3 }
4539   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4540 }

```

```

4541 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4542 {
4543   \int_step_inline:nn { \int_use:N \c@iRow }

```

```

4544 {
4545   \int_step_inline:nn { \int_use:N \c@jCol }
4546   {
4547     \int_if_even:nTF { ####1 + #1 }
4548     { \@@_cellcolor [ #1 ] { #2 } }
4549     { \@@_cellcolor [ #1 ] { #3 } }
4550     { ##1 - ####1 }
4551   }
4552 }
4553 }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns) and the cells in the “corners”.

```

4554 \NewDocumentCommand \@@_arraycolor { 0 { } m }
4555 {
4556   \@@_rectanglecolor [ #1 ] { #2 }
4557   { 1 - 1 }
4558   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4559 }

4560 \keys_define:nn { NiceMatrix / rowcolors }
4561 {
4562   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4563   respect-blocks .default:n = true ,
4564   cols .tl_set:N = \l_@@_cols_tl ,
4565   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4566   restart .default:n = true ,
4567   unknown .code:n = \@@_error:n { Unknown~key-for-rowcolors }
4568 }

```

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*.

```

4569 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
4570 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

4571   \group_begin:
4572   \seq_clear_new:N \l_@@_colors_seq
4573   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
4574   \tl_clear_new:N \l_@@_cols_tl
4575   \tl_set:Nn \l_@@_cols_tl { - }
4576   \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).

```

4577   \int_zero_new:N \l_@@_color_int
4578   \int_set:Nn \l_@@_color_int 1
4579   \bool_if:NT \l_@@_respect_blocks_bool
4580   {

```

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`).

```

4581   \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4582   \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
4583   { \@@_not_in_exterior_p:nnnnn #1 }
4584 }

```

```

4585 \pgfpicture
4586 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

4587 \clist_map_inline:nn { #2 }
4588 {
4589   \tl_set:Nn \l_tmpa_tl { ##1 }
4590   \tl_if_in:NnTF \l_tmpa_tl { - }
4591   { \@@_cut_on_hyphen:w ##1 \q_stop }
4592   { \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.

```

4593   \int_set:Nn \l_tmpa_int \l_tmpa_tl
4594   \bool_if:NNTF \l_@@_rowcolors_restart_bool
4595   { \int_set:Nn \l_@@_color_int 1 }
4596   { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
4597   \int_zero_new:N \l_@@_tmpc_int
4598   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
4599   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
4600   {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4601     \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).

```

4602     \bool_if:NT \l_@@_respect_blocks_bool
4603     {
4604       \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
4605       { \@@_intersect_our_row_p:nnnnn ###1 }
4606       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4607     }
4608     \tl_set:Nx \l_@@_rows_tl
4609     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

4610     \tl_clear_new:N \l_@@_color_tl
4611     \tl_set:Nx \l_@@_color_tl
4612     {
4613       \@@_color_index:n
4614       {
4615         \int_mod:nn
4616         { \l_@@_color_int - 1 }
4617         { \seq_count:N \l_@@_colors_seq }
4618         + 1
4619       }
4620     }
4621     \tl_if_empty:NNTF \l_@@_color_tl
4622     {
4623       \@@_add_to_colors_seq:xx
4624       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
4625       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
4626     }
4627     \int_incr:N \l_@@_color_int
4628     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4629   }
4630 }
4631 \endpgfpicture
4632 \group_end:
4633 }

```

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.

```

4634 \cs_new:Npn \@@_color_index:n #1

```

```

4635 {
4636   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
4637   { \@@_color_index:n { #1 - 1 } }
4638   { \seq_item:Nn \l_@@_colors_seq { #1 } }
4639 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

4640 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4641 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

4642 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
4643 {
4644   \int_compare:nNt { #3 } > \l_tmpb_int
4645   { \int_set:Nn \l_tmpb_int { #3 } }
4646 }

4647 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
4648 {
4649   \bool_lazy_or:nnTF
4650   { \int_compare_p:nNn { #4 } = \c_zero_int }
4651   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
4652   \prg_return_false:
4653   \prg_return_true:
4654 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

4655 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
4656 {
4657   \bool_if:nTF
4658   {
4659     \int_compare_p:n { #1 <= \l_tmpa_int }
4660     &&
4661     \int_compare_p:n { \l_tmpa_int <= #3 }
4662   }
4663   \prg_return_true:
4664   \prg_return_false:
4665 }

```

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`).

```

4666 \cs_new_protected:Npn \@@_cartesian_path:n #1
4667 {
4668   \bool_lazy_and:nnT
4669   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4670   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4671   {
4672     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4673     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4674   }

```

We begin the loop over the columns.

```

4675   \clist_map_inline:Nn \l_@@_cols_tl
4676   {
4677     \tl_set:Nn \l_tmpa_tl { ##1 }
4678     \tl_if_in:NnTF \l_tmpa_tl { - }

```



```

4679     { \@@_cut_on_hyphen:w ##1 \q_stop }
4680     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4681 \bool_lazy_or:nnT
4682     { \tl_if_blank_p:V \l_tmpa_tl }
4683     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4684     { \tl_set:Nn \l_tmpa_tl { 1 } }
4685 \bool_lazy_or:nnT
4686     { \tl_if_blank_p:V \l_tmpb_tl }
4687     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4688     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4689 \int_compare:nNnT \l_tmpb_tl > \c@jCol
4690     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

4691     \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.

```

4692     \@@_qpoint:n { col - \l_tmpa_tl }
4693     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4694     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4695     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4696     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
4697     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4698     \clist_map_inline:Nn \l_@@_rows_tl
4699     {
4700         \tl_set:Nn \l_tmpa_tl { #####1 }
4701         \tl_if_in:NnTF \l_tmpa_tl { - }
4702         { \@@_cut_on_hyphen:w #####1 \q_stop }
4703         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4704         \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4705         \tl_if_empty:NT \l_tmpb_tl
4706         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4707         \int_compare:nNnT \l_tmpb_tl > \c@iRow
4708         { \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`.

```

4709         \seq_if_in:NxF \l_@@_corners_cells_seq
4710         { \l_tmpa_tl - \l_@@_tmpc_tl }
4711         {
4712             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
4713             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4714             \@@_qpoint:n { row - \l_tmpa_tl }
4715             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4716             \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4717             \pgfpathrectanglecorners
4718             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
4719             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4720         }
4721     }
4722 }
4723 }

```

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`).

```

4724 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to 2,4-6,8-* and `\c@jCol` equal to 10, the `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

4725 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4726 {
4727   \clist_set_eq:NN \l_tmpa_clist #1
4728   \clist_clear:N #1
4729   \clist_map_inline:Nn \l_tmpa_clist
4730   {
4731     \tl_set:Nn \l_tmpa_tl { ##1 }
4732     \tl_if_in:NnTF \l_tmpa_tl { - }
4733     { \@@_cut_on_hyphen:w ##1 \q_stop }
4734     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4735     \bool_lazy_or:nnT
4736     { \tl_if_blank_p:V \l_tmpa_tl }
4737     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4738     { \tl_set:Nn \l_tmpa_tl { 1 } }
4739     \bool_lazy_or:nnT
4740     { \tl_if_blank_p:V \l_tmpb_tl }
4741     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4742     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4743     \int_compare:nNnT \l_tmpb_tl > #2
4744     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4745     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4746     { \clist_put_right:Nn #1 { #####1 } }
4747   }
4748 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```

4749 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4750 {
4751   \peek_remove_spaces:n
4752   {
4753     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4754     {

```

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`).

```

4755       \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
4756       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4757     }
4758   }
4759 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the tabular.

```

4760 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4761 {
4762   \peek_remove_spaces:n
4763   {
4764     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4765     {
4766       \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4767       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4768       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4769     }
4770   }
4771 }

```

```

4772 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4773 {

```

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).

```
4774 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4775 {
```

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).

```
4776 \tl_gput_left:Nx \g_nicematrix_code_before_tl
4777 {
4778   \exp_not:N \columncolor [ #1 ]
4779   { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4780 }
4781 }
4782 }
```

The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
4783 \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`.

```
4784 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4785 {
4786   \int_compare:nNnTF \l_@@_first_col_int = 0
4787   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4788   {
4789     \int_compare:nNnTF \c@jCol = 0
4790     {
4791       \int_compare:nNnF \c@iRow = { -1 }
4792       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4793     }
4794     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4795   }
4796 }
```

This definition may seem complicated but we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell.

The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```
4797 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4798 {
4799   \int_compare:nNnF \c@iRow = 0
4800   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4801 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to `-2` or `-1` (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or “subsystem” of nicematrix wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```
4802 \keys_define:nn { NiceMatrix / Rules }
4803 {
4804   position .int_set:N = \l_@@_position_int ,
4805   position .value_required:n = true ,
4806   start .int_set:N = \l_@@_start_int ,
4807   start .initial:n = 1 ,
4808   end .int_set:N = \l_@@_end_int ,
4809 }
```

It’s possible that the rule won’t be drawn continuously from `start` ot `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```
4810 \keys_define:nn { NiceMatrix / RulesBis }
4811 {
4812   multiplicity .int_set:N = \l_@@_multiplicity_int ,
4813   multiplicity .initial:n = 1 ,
4814   dotted .bool_set:N = \l_@@_dotted_bool ,
4815   dotted .initial:n = false ,
4816   dotted .default:n = true ,
4817   color .code:n = \@@_set_CT@arc@: #1 \q_stop ,
4818   color .value_required:n = true ,
4819   sep-color .code:n = \@@_set_CT@drsc@: #1 \q_stop ,
4820   sep-color .value_required:n = true ,
```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with Tikz.

```
4821   tikz .tl_set:N = \l_@@_tikz_rule_tl ,
4822   tikz .value_required:n = true ,
4823   tikz .initial:n = ,
4824   total-width .dim_set:N = \l_@@_rule_width_dim ,
4825   total-width .value_required:n = true ,
4826   width .meta:n = { total-width = #1 }
4827 }
```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of *key=value* pairs.

```
4828 \cs_new_protected:Npn \@@_vline:n #1
4829 {
```

The group is for the options.

```
4830   \group_begin:
4831   \int_zero_new:N \l_@@_end_int
4832   \int_set_eq:NN \l_@@_end_int \c@iRow
4833   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```
4834   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
4835     \@@_vline_i:
4836   \group_end:
4837 }
```

```

4838 \cs_new_protected:Npn \l_@@_vline_i:
4839 {
4840   \int_zero_new:N \l_@@_local_start_int
4841   \int_zero_new:N \l_@@_local_end_int

```

\l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. When we have found a row corresponding to a rule to draw, we note its number in \l_@@_tmpc_tl.

```

4842   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
4843   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
4844     \l_tmpa_tl
4845   {

```

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.

```

4846     \bool_gset_true:N \g_tmpa_bool
4847     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4848       { \@@_test_vline_in_block:nnnnn ##1 }
4849     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4850       { \@@_test_vline_in_block:nnnnn ##1 }
4851     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4852       { \@@_test_vline_in_stroken_block:nnnn ##1 }
4853     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
4854     \bool_if:NTF \g_tmpa_bool
4855       {
4856         \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. \l_@@_local_start_int will be the starting row of the rule that we will have to draw.

```

4857         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
4858       }
4859     {
4860       \int_compare:nNnT \l_@@_local_start_int > 0
4861       {
4862         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
4863         \@@_vline_ii:
4864         \int_zero:N \l_@@_local_start_int
4865       }
4866     }
4867   }
4868   \int_compare:nNnT \l_@@_local_start_int > 0
4869   {
4870     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
4871     \@@_vline_ii:
4872   }
4873 }

```

```

4874 \cs_new_protected:Npn \@@_test_in_corner_v:
4875 {
4876   \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
4877   {
4878     \seq_if_in:NxT
4879       \l_@@_corners_cells_seq
4880       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
4881     { \bool_set_false:N \g_tmpa_bool }
4882   }
4883   {
4884     \seq_if_in:NxT
4885       \l_@@_corners_cells_seq
4886       { \l_tmpa_tl - \l_tmpb_tl }
4887     {
4888       \int_compare:nNnTF \l_tmpb_tl = 1

```

```

4889         { \bool_set_false:N \g_tmpa_bool }
4890         {
4891             \seq_if_in:NxT
4892             \l_@@_corners_cells_seq
4893             { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
4894             { \bool_set_false:N \g_tmpa_bool }
4895         }
4896     }
4897 }
4898 }

```

```

4899 \cs_new_protected:Npn \@@_vline_ii:
4900 {
4901     \bool_set_false:N \l_@@_dotted_bool
4902     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
4903     \bool_if:NTF \l_@@_dotted_bool
4904     \@@_vline_iv:
4905     {
4906         \tl_if_empty:NTF \l_@@_tikz_rule_tl
4907         \@@_vline_iii:
4908         \@@_vline_v:
4909     }
4910 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

4911 \cs_new_protected:Npn \@@_vline_iii:
4912 {
4913     \pgfpicture
4914     \pgfrememberpicturepositiononpagetrue
4915     \pgf@relevantforpicturesizefalse
4916     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4917     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4918     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4919     \dim_set:Nn \l_tmpb_dim
4920     {
4921         \pgf@x
4922         - 0.5 \l_@@_rule_width_dim
4923         +
4924         ( \arrayrulewidth * \l_@@_multiplicity_int
4925           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
4926     }
4927     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4928     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4929     \bool_lazy_all:nT
4930     {
4931         { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
4932         { \cs_if_exist_p:N \CT@drsc@ }
4933         { ! \tl_if_blank_p:V \CT@drsc@ }
4934     }
4935     {
4936         \group_begin:
4937         \CT@drsc@
4938         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4939         \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
4940         \dim_set:Nn \l_@@_tmpd_dim
4941         {
4942             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
4943             * ( \l_@@_multiplicity_int - 1 )
4944         }
4945         \pgfpathrectanglecorners
4946         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4947         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }

```

```

4948     \pgfusepath { fill }
4949     \group_end:
4950   }
4951   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4952   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
4953   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
4954   {
4955     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4956     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4957     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4958     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
4959   }
4960   \CT@arc@
4961   \pgfsetlinewidth { 1.1 \arrayrulewidth }
4962   \pgfsetrectcap
4963   \pgfusepathqstroke
4964   \endpgfpicture
4965 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

4966 \cs_new_protected:Npn \@@_vline_iv:
4967 {
4968   \pgfpicture
4969   \pgfrememberpicturepositiononpagetrue
4970   \pgf@relevantforpicturesizefalse
4971   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4972   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
4973   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4974   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4975   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4976   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4977   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4978   \CT@arc@
4979   \@@_draw_line:
4980   \endpgfpicture
4981 }

```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```

4982 \cs_new_protected:Npn \@@_vline_v:
4983 {
4984   \begin {tikzpicture }
4985   \pgfrememberpicturepositiononpagetrue
4986   \pgf@relevantforpicturesizefalse
4987   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4988   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4989   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4990   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
4991   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4992   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4993   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
4994   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
4995     ( \l_tmpb_dim , \l_tmpa_dim ) --
4996     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
4997   \end {tikzpicture }
4998 }

```

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).

```

4999 \cs_new_protected:Npn \@@_draw_vlines:
5000 {
5001   \int_step_inline:nnn

```

```

5002     {
5003       \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5004         1 2
5005     }
5006     {
5007       \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5008         { \int_eval:n { \c@jCol + 1 } }
5009         \c@jCol
5010     }
5011     {
5012       \tl_if_eq:NnF \l_@@_vlines_clist { all }
5013       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5014       { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5015     }
5016   }

```

The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs of the form `{NiceMatrix/Rules}`.

```

5017 \cs_new_protected:Npn \@@_hline:n #1
5018   {

```

The group is for the options.

```

5019   \group_begin:
5020   \int_zero_new:N \l_@@_end_int
5021   \int_set_eq:NN \l_@@_end_int \c@jCol
5022   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5023   \@@_hline_i:
5024   \group_end:
5025 }
5026 \cs_new_protected:Npn \@@_hline_i:
5027   {
5028     \int_zero_new:N \l_@@_local_start_int
5029     \int_zero_new:N \l_@@_local_end_int

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a column corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

5030   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5031   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5032     \l_tmpb_tl
5033   {

```

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.

```

5034     \bool_gset_true:N \g_tmpa_bool
5035     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5036       { \@@_test_hline_in_block:nnnnn ##1 }
5037     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5038       { \@@_test_hline_in_block:nnnnn ##1 }
5039     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5040       { \@@_test_hline_in_stroken_block:nnnn ##1 }
5041     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5042     \bool_if:NTF \g_tmpa_bool
5043       {
5044         \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```

5045       { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5046     }

```



```

5047         {
5048             \int_compare:nNnT \l_@@_local_start_int > 0
5049             {
5050                 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5051                 \@@_hline_ii:
5052                 \int_zero:N \l_@@_local_start_int
5053             }
5054         }
5055     }
5056     \int_compare:nNnT \l_@@_local_start_int > 0
5057     {
5058         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5059         \@@_hline_ii:
5060     }
5061 }

5062 \cs_new_protected:Npn \@@_test_in_corner_h:
5063 {
5064     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5065     {
5066         \seq_if_in:NxT
5067         \l_@@_corners_cells_seq
5068         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5069         { \bool_set_false:N \g_tmpa_bool }
5070     }
5071     {
5072         \seq_if_in:NxT
5073         \l_@@_corners_cells_seq
5074         { \l_tmpa_tl - \l_tmpb_tl }
5075         {
5076             \int_compare:nNnTF \l_tmpa_tl = 1
5077             { \bool_set_false:N \g_tmpa_bool }
5078             {
5079                 \seq_if_in:NxT
5080                 \l_@@_corners_cells_seq
5081                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5082                 { \bool_set_false:N \g_tmpa_bool }
5083             }
5084         }
5085     }
5086 }

5087 \cs_new_protected:Npn \@@_hline_ii:
5088 {
5089     \bool_set_false:N \l_@@_dotted_bool
5090     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5091     \bool_if:NTF \l_@@_dotted_bool
5092     \@@_hline_iv:
5093     {
5094         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5095         \@@_hline_iii:
5096         \@@_hline_v:
5097     }
5098 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

5099 \cs_new_protected:Npn \@@_hline_iii:
5100 {
5101     \pgfpicture
5102     \pgfrememberpicturepositiononpagetrue
5103     \pgf@relevantforpicturesizefalse

```

```

5104 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5105 \dim_set_eq:NN \l_tmpa_dim \pgf@x
5106 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5107 \dim_set:Nn \l_tmpb_dim
5108 {
5109   \pgf@y
5110   - 0.5 \l_@@_rule_width_dim
5111   +
5112   ( \arrayrulewidth * \l_@@_multiplicity_int
5113     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5114 }
5115 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5116 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5117 \bool_lazy_all:nT
5118 {
5119   { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5120   { \cs_if_exist_p:N \CT@drsc@ }
5121   { ! \tl_if_blank_p:V \CT@drsc@ }
5122 }
5123 {
5124   \group_begin:
5125   \CT@drsc@
5126   \dim_set:Nn \l_@@_tmpd_dim
5127   {
5128     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5129     * ( \l_@@_multiplicity_int - 1 )
5130   }
5131   \pgfpathrectanglecorners
5132   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5133   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5134   \pgfusepathqfill
5135   \group_end:
5136 }
5137 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5138 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5139 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5140 {
5141   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5142   \dim_sub:Nn \l_tmpb_dim \doublerulesep
5143   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5144   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5145 }
5146 \CT@arc@
5147 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5148 \pgfsetrectcap
5149 \pgfusepathqstroke
5150 \endpgfpicture
5151 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

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}

```

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

```

5152 \cs_new_protected:Npn \@@_hline_iv:
5153 {
5154   \pgfpicture
5155   \pgfrememberpicturepositiononpagetrue
5156   \pgf@relevantforpicturesizefalse
5157   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5158   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5159   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
5160   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5161   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5162   \int_compare:nNnT \l_@@_local_start_int = 1
5163   {
5164     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5165     \bool_if:NT \l_@@_NiceArray_bool
5166     { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by $0.5 \l_@@_xdots_inter_dim$ is *ad hoc* for a better result.

```

5167   \tl_if_eq:NnF \g_@@_left_delim_tl (
5168     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
5169   )
5170   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5171   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5172   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5173   {
5174     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5175     \bool_if:NT \l_@@_NiceArray_bool
5176     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5177     \tl_if_eq:NnF \g_@@_right_delim_tl )
5178     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
5179   }
5180   \CT@arc@
5181   \@@_draw_line:
5182   \endpgfpicture
5183 }

```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```

5184 \cs_new_protected:Npn \@@_hline_v:
5185 {
5186   \begin { tikzpicture }
5187   \pgfrememberpicturepositiononpagetrue
5188   \pgf@relevantforpicturesizefalse
5189   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5190   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5191   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5192   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5193   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5194   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5195   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5196   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5197     ( \l_tmpa_dim , \l_tmpb_dim ) --
5198     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5199   \end { tikzpicture }
5200 }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners (if the key `corners` is used)).

```

5201 \cs_new_protected:Npn \@@_draw_hlines:
5202 {
5203   \int_step_inline:nnn
5204   {
5205     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5206     1 2
5207   }
5208   {
5209     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5210     { \int_eval:n { \c@iRow + 1 } }
5211     \c@iRow
5212   }
5213   {
5214     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5215     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5216     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
5217   }
5218 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5219 \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.

```

5220 \cs_set:Npn \@@_Hline_i:n #1
5221 {
5222   \peek_remove_spaces:n
5223   {
5224     \peek_meaning:NTF \Hline
5225     { \@@_Hline_ii:nn { #1 + 1 } }
5226     { \@@_Hline_iii:n { #1 } }
5227   }
5228 }
5229 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5230 \cs_set:Npn \@@_Hline_iii:n #1
5231 {
5232   \skip_vertical:n
5233   {
5234     \arrayrulewidth * ( #1 )
5235     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
5236   }
5237   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5238   {
5239     \@@_hline:n
5240     {
5241       position = \int_eval:n { \c@iRow + 1 } ,
5242       multiplicity = #1
5243     }
5244   }
5245   \ifnum 0 = ` { \fi }
5246 }

```

Customized rules defined by the final user

The final user can define a customized rule by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of `key=value` pairs.

Among the keys available in that list, there is the key `letter` to specify a letter that the final user will use in the preamble of the array. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix / ColumnTypes}`. That set of keys is used

to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```
5247 \keys_define:nn { NiceMatrix / ColumnTypes } { }
```

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```
5248 \cs_new_protected:Npn \@@_custom_line:n #1
5249 {
5250   \str_clear_new:N \l_@@_command_str
5251   \str_clear_new:N \l_@@_letter_str
5252   \keys_set_known:nnN { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl
```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```
5253   \bool_lazy_and:nnTF
5254     { \str_if_empty_p:N \l_@@_letter_str }
5255     { \str_if_empty_p:N \l_@@_command_str }
5256     { \@@_error:n { No~letter~and~no~command } }
5257     { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
5258 }

5259 \keys_define:nn { NiceMatrix / custom-line }
5260 {
5261   % here, we will use change in the future to use .str_set:N
5262   letter .code:n = \str_set:Nn \l_@@_letter_str { #1 } ,
5263   letter .value_required:n = true ,
5264   % here, we will use change in the future to use .str_set:N
5265   command .code:n = \str_set:Nn \l_@@_command_str { #1 } ,
5266   command .value_required:n = true ,
5267 }
```

```
5268 \cs_new_protected:Npn \@@_custom_line_i:n #1
5269 {
```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```
5270   \bool_set_false:N \l_@@_tikz_rule_bool
5271   \bool_set_false:N \l_@@_dotted_rule_bool
5272   \bool_set_false:N \l_@@_color_bool

5273   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
5274   \bool_if:NT \l_@@_tikz_rule_bool
5275   {
```

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.

```
5276     \cs_if_exist:NF \tikzpicture
5277     { \@@_error:n { tikz~in~custom-line~without~tikz } }
5278     \bool_if:NT \l_@@_color_bool
5279     { \@@_error:n { color~in~custom-line~with~tikz } }
5280   }
5281   \bool_if:nT
5282   {
5283     \int_compare_p:nNn \l_@@_multiplicity_int > 1
5284     && \l_@@_dotted_rule_bool
5285   }
5286   { \@@_error:n { key~multiplicity~with~dotted } }
5287   \str_if_empty:NF \l_@@_letter_str
5288   {
5289     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
5290     { \@@_error:n { Several~letters } }
```

```

5291     {
5292         \exp_args:NnV \tl_if_in:NnTF
5293         \c_@@_forbidden_letters_str \l_@@_letter_str
5294         { \@@_error:n { Forbidden~letter } }
5295     }

```

The final user can, locally, redefine a letter of column type. That's compatible with the use of `\keys_define:nn`: the definition is local and may overwrite a previous definition.

```

5296         \keys_define:nx { NiceMatrix / ColumnTypes }
5297         {
5298             \l_@@_letter_str .code:n =
5299             { \@@_v_custom_line:n { \exp_not:n { #1 } } }
5300         }
5301     }
5302 }
5303 }
5304 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
5305 }
5306 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

5307 \keys_define:nn { NiceMatrix / custom-line-bis }
5308 {
5309     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5310     multiplicity .initial:n = 1 ,
5311     multiplicity .value_required:n = true ,
5312     color .code:n = \bool_set_true:N \l_@@_color_bool ,
5313     color .value_required:n = true ,
5314     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5315     tikz .value_required:n = true ,
5316     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5317     dotted .value_forbidden:n = true ,
5318     total-width .code:n = { } ,
5319     total-width .value_required:n = true ,
5320     width .code:n = { } ,
5321     width .value_required:n = true ,
5322     sep-color .code:n = { } ,
5323     sep-color .value_required:n = true ,
5324     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
5325 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

5326 \bool_new:N \l_@@_dotted_rule_bool
5327 \bool_new:N \l_@@_tikz_rule_bool
5328 \bool_new:N \l_@@_color_bool

```

The following keys are used to determine the total width of the line (including the spaces on both sides of the line). The key `width` is deprecated and has been replaced by the key `total-width`.

```

5329 \keys_define:nn { NiceMatrix / custom-line-width }
5330 {
5331     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5332     multiplicity .initial:n = 1 ,
5333     multiplicity .value_required:n = true ,
5334     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
5335     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5336             \bool_set_true:N \l_@@_total_width_bool ,
5337     total-width .value_required:n = true ,
5338     width .meta:n = { total-width = #1 } ,

```

```

5339     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
5340 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name). #1 is the whole set of keys to pass to \@@_line:n.

```

5341 \cs_new_protected:Npn \@@_h_custom_line:n #1
5342 {

```

We use \cs_set:cpn and not \cs_new:cpn because we want a local definition. Moreover, the command must *not* be protected since it begins with \noalign.

```

5343     \cs_set:cpn { nicematrix - \l_@@_command_str }
5344     {
5345         \noalign
5346         {
5347             \@@_compute_rule_width:n { #1 }
5348             \skip_vertical:n { \l_@@_rule_width_dim }
5349             \tl_gput_right:Nx \g_@@_internal_code_after_tl
5350             {
5351                 \@@_hline:n
5352                 {
5353                     #1 ,
5354                     position = \int_eval:n { \c@iRow + 1 } ,
5355                     total-width = \dim_use:N \l_@@_rule_width_dim
5356                 }
5357             }
5358         }
5359     }
5360     \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
5361 }
5362 \cs_generate_variant:Nn \@@_h_custom_line:nn { n V }
5363 \cs_new_protected:Npn \@@_compute_rule_width:n #1
5364 {
5365     \bool_set_false:N \l_@@_tikz_rule_bool
5366     \bool_set_false:N \l_@@_total_width_bool
5367     \bool_set_false:N \l_@@_dotted_rule_bool
5368     \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
5369     \bool_if:NF \l_@@_total_width_bool
5370     {
5371         \bool_if:NTF \l_@@_dotted_rule_bool
5372         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
5373         {
5374             \bool_if:NF \l_@@_tikz_rule_bool
5375             {
5376                 \dim_set:Nn \l_@@_rule_width_dim
5377                 {
5378                     \arrayrulewidth * \l_@@_multiplicity_int
5379                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
5380                 }
5381             }
5382         }
5383     }
5384 }
5385 \cs_new_protected:Npn \@@_v_custom_line:n #1
5386 {
5387     \@@_compute_rule_width:n { #1 }

```

In the following line, the \dim_use:N is mandatory since we do an expansion.

```

5388     \tl_gput_right:Nx \g_@@_preamble_tl
5389     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
5390     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5391     {
5392         \@@_vline:n

```

```

5393     {
5394         #1 ,
5395         position = \int_eval:n { \c@jCol + 1 } ,
5396         total-width = \dim_use:N \l_@@_rule_width_dim
5397     }
5398 }
5399 }
5400 \@@_custom_line:n { letter = : , command = hdottedline , dotted }

```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to false.

```

5401 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
5402 {
5403     \bool_lazy_all:nT
5404     {
5405         { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
5406         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5407         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5408         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5409     }
5410     { \bool_gset_false:N \g_tmpa_bool }
5411 }

```

The same for vertical rules.

```

5412 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
5413 {
5414     \bool_lazy_all:nT
5415     {
5416         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5417         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5418         { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
5419         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5420     }
5421     { \bool_gset_false:N \g_tmpa_bool }
5422 }
5423 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
5424 {
5425     \bool_lazy_all:nT
5426     {
5427         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5428         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
5429         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5430         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5431     }
5432     { \bool_gset_false:N \g_tmpa_bool }
5433 }
5434 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
5435 {
5436     \bool_lazy_all:nT
5437     {
5438         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5439         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5440         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5441         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
5442     }
5443     { \bool_gset_false:N \g_tmpa_bool }
5444 }

```


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.

```
5445 \cs_new_protected:Npn \@@_compute_corners:
5446 {
```

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.

```
5447   \seq_clear_new:N \l_@@_corners_cells_seq
5448   \clist_map_inline:Nn \l_@@_corners_clist
5449   {
5450     \str_case:nnF { ##1 }
5451     {
5452       { NW }
5453       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
5454       { NE }
5455       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
5456       { SW }
5457       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
5458       { SE }
5459       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
5460     }
5461     { \@@_error:nn { bad~corner } { ##1 } }
5462   }
```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```
5463   \seq_if_empty:NF \l_@@_corners_cells_seq
5464   {
```

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.

```
5465     \tl_gput_right:Nx \g_@@_aux_tl
5466     {
5467       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
5468       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
5469     }
5470   }
5471 }
```

“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.

```
5472 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
5473 {
```

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.

```
5474   \bool_set_false:N \l_tmpa_bool
5475   \int_zero_new:N \l_@@_last_empty_row_int
5476   \int_set:Nn \l_@@_last_empty_row_int { #1 }
```

```

5477 \int_step_inline:nnnn { #1 } { #3 } { #5 }
5478 {
5479   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
5480   \bool_lazy_or:nnTF
5481   {
5482     \cs_if_exist_p:c
5483     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
5484   }
5485   \l_tmpb_bool
5486   { \bool_set_true:N \l_tmpa_bool }
5487   {
5488     \bool_if:NF \l_tmpa_bool
5489     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
5490   }
5491 }

```

Now, you determine the last empty cell in the row of number 1.

```

5492 \bool_set_false:N \l_tmpa_bool
5493 \int_zero_new:N \l_@@_last_empty_column_int
5494 \int_set:Nn \l_@@_last_empty_column_int { #2 }
5495 \int_step_inline:nnnn { #2 } { #4 } { #6 }
5496 {
5497   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
5498   \bool_lazy_or:nnTF
5499   \l_tmpb_bool
5500   {
5501     \cs_if_exist_p:c
5502     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
5503   }
5504   { \bool_set_true:N \l_tmpa_bool }
5505   {
5506     \bool_if:NF \l_tmpa_bool
5507     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
5508   }
5509 }

```

Now, we loop over the rows.

```

5510 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
5511 {

```

We treat the row number ##1 with another loop.

```

5512 \bool_set_false:N \l_tmpa_bool
5513 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
5514 {
5515   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
5516   \bool_lazy_or:nnTF
5517   \l_tmpb_bool
5518   {
5519     \cs_if_exist_p:c
5520     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
5521   }
5522   { \bool_set_true:N \l_tmpa_bool }
5523   {
5524     \bool_if:NF \l_tmpa_bool
5525     {
5526       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
5527       \seq_put_right:Nn
5528       \l_@@_corners_cells_seq
5529       { ##1 - #####1 }
5530     }
5531   }
5532 }
5533 }
5534 }

```

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`).

```

5535 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
5536 {
5537   \int_set:Nn \l_tmpa_int { #1 }
5538   \int_set:Nn \l_tmpb_int { #2 }
5539   \bool_set_false:N \l_tmpb_bool
5540   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5541     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
5542 }

5543 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
5544 {
5545   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
5546   {
5547     \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
5548     {
5549       \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
5550       {
5551         \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
5552         { \bool_set_true:N \l_tmpb_bool }
5553       }
5554     }
5555   }
5556 }

```

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.

```

5557 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

5558 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5559 {
5560   auto-columns-width .code:n =
5561   {
5562     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5563     \dim_gzero_new:N \g_@@_max_cell_width_dim
5564     \bool_set_true:N \l_@@_auto_columns_width_bool
5565   }
5566 }

5567 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
5568 {
5569   \int_gincr:N \g_@@_NiceMatrixBlock_int
5570   \dim_zero:N \l_@@_columns_width_dim
5571   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5572   \bool_if:NT \l_@@_block_auto_columns_width_bool
5573   {
5574     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5575     {
5576       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
5577         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5578     }
5579   }
5580 }

```

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`).

```

5581 {
5582   \bool_if:NT \l_@@_block_auto_columns_width_bool
5583   {
5584     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5585     \iow_shipout:Nx \@mainaux
5586     {
5587       \cs_gset:cpn
5588       { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
5589       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5590     }
5591     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5592   }
5593 }

```

The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

5594 \cs_generate_variant:Nn \dim_min:nn { v n }
5595 \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).

```

5596 \cs_new_protected:Npn \@@_create_extra_nodes:
5597 {
5598   \bool_if:nTF \l_@@_medium_nodes_bool
5599   {
5600     \bool_if:nTF \l_@@_large_nodes_bool
5601     \@@_create_medium_and_large_nodes:
5602     \@@_create_medium_nodes:
5603   }
5604   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5605 }

```

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`.

```

5606 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5607 {
5608   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:

```

```

5609 {
5610   \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
5611   \dim_set_eq:cn { l_@@_row\_@@_i: _min_dim } \c_max_dim
5612   \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
5613   \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
5614 }
5615 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5616 {
5617   \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
5618   \dim_set_eq:cn { l_@@_column\_@@_j: _min_dim } \c_max_dim
5619   \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
5620   \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
5621 }

```

We begin the two nested loops over the rows and the columns of the array.

```

5622 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5623 {
5624   \int_step_variable:nnNn
5625     \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.

```

5626 {
5627   \cs_if_exist:cT
5628     { 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.

```

5629 {
5630   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5631   \dim_set:cn { l_@@_row\_@@_i: _min_dim }
5632   { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
5633   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5634   {
5635     \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
5636     { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
5637   }

```

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.

```

5638   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
5639   \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
5640   { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
5641   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5642   {
5643     \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
5644     { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
5645   }
5646 }
5647 }
5648 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

5649 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5650 {
5651   \dim_compare:nnNT
5652     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
5653   {
5654     \@@_qpoint:n { row - \@@_i: - base }
5655     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
5656     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
5657   }
5658 }
5659 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

```

5660 {
5661   \dim_compare:nNnT
5662     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
5663     {
5664       \@@_qpoint:n { col - \@@_j: }
5665       \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
5666       \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
5667     }
5668   }
5669 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

5670 \cs_new_protected:Npn \@@_create_medium_nodes:
5671 {
5672   \pgfpicture
5673   \pgfrememberpicturepositiononpagetrue
5674   \pgf@relevantforpicturesizefalse
5675   \@@_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”.

```

5676   \tl_set:Nn \l_@@_suffix_tl { -medium }
5677   \@@_create_nodes:
5678   \endpgfpicture
5679 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁷¹. 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:`.

```

5680 \cs_new_protected:Npn \@@_create_large_nodes:
5681 {
5682   \pgfpicture
5683   \pgfrememberpicturepositiononpagetrue
5684   \pgf@relevantforpicturesizefalse
5685   \@@_computations_for_medium_nodes:
5686   \@@_computations_for_large_nodes:
5687   \tl_set:Nn \l_@@_suffix_tl { - large }
5688   \@@_create_nodes:
5689   \endpgfpicture
5690 }
5691 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5692 {
5693   \pgfpicture
5694   \pgfrememberpicturepositiononpagetrue
5695   \pgf@relevantforpicturesizefalse
5696   \@@_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”.

```

5697   \tl_set:Nn \l_@@_suffix_tl { - medium }
5698   \@@_create_nodes:
5699   \@@_computations_for_large_nodes:
5700   \tl_set:Nn \l_@@_suffix_tl { - large }
5701   \@@_create_nodes:
5702   \endpgfpicture
5703 }

```

⁷¹If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

5704 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5705 {
5706   \int_set:Nn \l_@@_first_row_int 1
5707   \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`.

```

5708   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5709   {
5710     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5711     {
5712       (
5713         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5714         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
5715       )
5716       / 2
5717     }
5718     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
5719     { l_@@_row _ \@@_i: _ min _ dim }
5720   }
5721   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5722   {
5723     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5724     {
5725       (
5726         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5727         \dim_use:c
5728         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
5729       )
5730       / 2
5731     }
5732     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
5733     { l_@@_column _ \@@_j: _ max _ dim }
5734   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

5735   \dim_sub:cn
5736   { l_@@_column _ 1 _ min _ dim }
5737   \l_@@_left_margin_dim
5738   \dim_add:cn
5739   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5740   \l_@@_right_margin_dim
5741 }

```

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).

```

5742 \cs_new_protected:Npn \@@_create_nodes:
5743 {
5744   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5745   {
5746     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5747     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

5748       \@@_pgf_rect_node:nnnnn
5749       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5750       { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }

```

```

5751         { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
5752         { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
5753         { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
5754     \str_if_empty:NF \l_@@_name_str
5755     {
5756         \pgfnodealias
5757         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5758         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5759     }
5760 }
5761 }

```

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 .

```

5762     \seq_mapthread_function:NNN
5763     \g_@@_multicolumn_cells_seq
5764     \g_@@_multicolumn_sizes_seq
5765     \@@_node_for_multicolumn:nn
5766 }

5767 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5768 {
5769     \cs_set_nopar:Npn \@@_i: { #1 }
5770     \cs_set_nopar:Npn \@@_j: { #2 }
5771 }

```

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”).

```

5772 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5773 {
5774     \@@_extract_coords_values: #1 \q_stop
5775     \@@_pgf_rect_node:nnnnn
5776     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5777     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
5778     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
5779     { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
5780     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
5781     \str_if_empty:NF \l_@@_name_str
5782     {
5783         \pgfnodealias
5784         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5785         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
5786     }
5787 }

```

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.

```

5788 \keys_define:nn { NiceMatrix / Block / FirstPass }
5789 {
5790     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5791     l .value_forbidden:n = true ,
5792     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5793     r .value_forbidden:n = true ,

```



```

5794   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5795   c .value_forbidden:n = true ,
5796   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5797   L .value_forbidden:n = true ,
5798   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5799   R .value_forbidden:n = true ,
5800   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5801   C .value_forbidden:n = true ,
5802   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5803   t .value_forbidden:n = true ,
5804   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5805   b .value_forbidden:n = true ,
5806   color .tl_set:N = \l_@@_color_tl ,
5807   color .value_required:n = true ,
5808   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
5809   respect-arraystretch .default:n = true ,
5810 }

```

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.

```

5811 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
5812 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not been provided by the user, you use `1-1` (that is to say a block of only one cell).

```

5813   \peek_remove_spaces:n
5814   {
5815     \tl_if_blank:nTF { #2 }
5816     { \@@_Block_i 1-1 \q_stop }
5817     { \@@_Block_i #2 \q_stop }
5818     { #1 } { #3 } { #4 }
5819   }
5820 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5821 \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` pairs, `#4` are the tokens to put before the math mode and the beginning of the small array of the block and `#5` is the label of the block.

```

5822 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5823 {

```

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).

```

5824   \bool_lazy_or:nnTF
5825   { \tl_if_blank_p:n { #1 } }
5826   { \str_if_eq_p:nn { #1 } { * } }
5827   { \int_set:Nn \l_tmpa_int { 100 } }
5828   { \int_set:Nn \l_tmpa_int { #1 } }
5829   \bool_lazy_or:nnTF
5830   { \tl_if_blank_p:n { #2 } }
5831   { \str_if_eq_p:nn { #2 } { * } }
5832   { \int_set:Nn \l_tmpb_int { 100 } }
5833   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5834 \int_compare:nNnTF \l_tmpb_int = 1
5835 {
5836   \str_if_empty:NTF \l_@@_hpos_cell_str
5837   { \str_set:Nn \l_@@_hpos_block_str c }
5838   { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
5839 }
5840 { \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.

```

5841 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
5842 \tl_set:Nx \l_tmpa_tl
5843 {
5844   { \int_use:N \c@iRow }
5845   { \int_use:N \c@jCol }
5846   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5847   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5848 }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

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

If the block is mono-column or mono-row, we have a special treatment. That’s why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by curryfication).

```

5849 \bool_if:nTF
5850 {
5851   (
5852     \int_compare_p:nNn { \l_tmpa_int } = 1
5853     ||
5854     \int_compare_p:nNn { \l_tmpb_int } = 1
5855   )
5856   && ! \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.

```

5857   && ! \l_@@_X_column_bool
5858 }
5859 { \exp_args:Nxx \@@_Block_iv:nnnnn }
5860 { \exp_args:Nxx \@@_Block_v:nnnnn }
5861 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5862 }

```

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).

```

5863 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5864 {
5865   \int_gincr:N \g_@@_block_box_int
5866   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5867   {
5868     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5869     {
5870       \@@_actually_diagbox:nnnnnn
5871       { \int_use:N \c@iRow }

```

```

5872         { \int_use:N \c@jCol }
5873         { \int_eval:n { \c@iRow + #1 - 1 } }
5874         { \int_eval:n { \c@jCol + #2 - 1 } }
5875         { \exp_not:n { ##1 } } } { \exp_not:n { ##2 } }
5876     }
5877 }
5878 \box_gclear_new:c
5879 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5880 \hbox_gset:cn
5881 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5882 {

```

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}`).

```

5883     \tl_if_empty:NTF \l_@@_color_tl
5884     { \int_compare:nNnT { #2 } = 1 \set@color }
5885     { \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`.

```

5886     \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5887     \group_begin:
5888     \bool_if:NF \l_@@_respect_arraystretch_bool
5889     { \cs_set:Npn \arraystretch { 1 } }
5890     \dim_zero:N \extrarowheight
5891     #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).

```

5892     \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
5893     \bool_if:NTF \l_@@_NiceTabular_bool
5894     {
5895         \bool_lazy_all:nTF
5896         {
5897             { \int_compare_p:nNn { #2 } = 1 }
5898             { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
5899             { ! \l_@@_respect_arraystretch_bool }
5900         }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

5901     {
5902         \begin { minipage } [ \l_@@_vpos_of_block_tl ]
5903         { \l_@@_col_width_dim }
5904         \str_case:Nn \l_@@_hpos_block_str
5905         {
5906             c \centering
5907             r \raggedleft
5908             l \raggedright
5909         }
5910         #5
5911         \end { minipage }
5912     }
5913     {
5914         \use:x
5915         {
5916             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5917             { @ { } \l_@@_hpos_block_str @ { } }
5918         }

```

```

5919         #5
5920     \end { tabular }
5921 }
5922 }
5923 {
5924     \c_math_toggle_token
5925     \use:x
5926     {
5927         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5928         { @ { } \l_@@_hpos_block_str @ { } }
5929     }
5930     #5
5931     \end { array }
5932     \c_math_toggle_token
5933 }
5934 \group_end:
5935 }
5936 \bool_if:NT \g_@@_rotate_bool
5937 {
5938     \box_grotate:cn
5939     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5940     { 90 }
5941     \bool_gset_false:N \g_@@_rotate_bool
5942 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5943 \int_compare:nNnT { #2 } = 1
5944 {
5945     \dim_gset:Nn \g_@@_blocks_wd_dim
5946     {
5947         \dim_max:nn
5948         \g_@@_blocks_wd_dim
5949         {
5950             \box_wd:c
5951             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5952         }
5953     }
5954 }

```

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.

```

5955 \int_compare:nNnT { #1 } = 1
5956 {
5957     \dim_gset:Nn \g_@@_blocks_ht_dim
5958     {
5959         \dim_max:nn
5960         \g_@@_blocks_ht_dim
5961         {
5962             \box_ht:c
5963             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5964         }
5965     }
5966     \dim_gset:Nn \g_@@_blocks_dp_dim
5967     {
5968         \dim_max:nn
5969         \g_@@_blocks_dp_dim
5970         {
5971             \box_dp:c
5972             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5973         }
5974     }
5975 }
5976 \seq_gput_right:Nx \g_@@_blocks_seq

```

```

5977 {
5978   \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.
5979 { \exp_not:n { #3 } , \l_@@_hpos_block_str }
5980 {
5981   \box_use_drop:c
5982   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5983 }
5984 }
5985 }

```

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.

```

5986 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5987 {
5988   \seq_gput_right:Nx \g_@@_blocks_seq
5989   {
5990     \l_tmpa_tl
5991     { \exp_not:n { #3 } }
5992     \exp_not:n
5993     {
5994       {
5995         \bool_if:NTF \l_@@_NiceTabular_bool
5996         {
5997           \group_begin:
5998           \bool_if:NF \l_@@_respect_arraystretch_bool
5999           { \cs_set:Npn \arraystretch { 1 } }
6000           \dim_zero:N \extrarowheight
6001           #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).

```

6002   \bool_if:NT \g_@@_rotate_bool
6003   { \str_set:Nn \l_@@_hpos_block_str c }
6004   \use:x
6005   {
6006     \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
6007     { @ { } \l_@@_hpos_block_str @ { } }
6008   }
6009   #5
6010   \end { tabular }
6011   \group_end:
6012 }
6013 {
6014   \group_begin:
6015   \bool_if:NF \l_@@_respect_arraystretch_bool
6016   { \cs_set:Npn \arraystretch { 1 } }
6017   \dim_zero:N \extrarowheight
6018   #4
6019   \bool_if:NT \g_@@_rotate_bool
6020   { \str_set:Nn \l_@@_hpos_block_str c }
6021   \c_math_toggle_token
6022   \use:x
6023   {
6024     \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]

```

```

6025         { @ { } \l_@@_hpos_block_str @ { } }
6026     }
6027     #5
6028     \end { array }
6029     \c_math_toggle_token
6030     \group_end:
6031 }
6032 }
6033 }
6034 }
6035 }

```

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).

```

6036 \keys_define:nn { NiceMatrix / Block / SecondPass }
6037 {
6038     tikz .code:n =
6039         \bool_if:NTF \c_@@_tikz_loaded_bool
6040         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
6041         { \@@_error:n { tikz-key-without~tikz } } ,
6042     tikz .value_required:n = true ,
6043     fill .tl_set:N = \l_@@_fill_tl ,
6044     fill .value_required:n = true ,
6045     draw .tl_set:N = \l_@@_draw_tl ,
6046     draw .default:n = default ,
6047     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6048     rounded-corners .default:n = 4 pt ,
6049     color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
6050     color .value_required:n = true ,
6051     borders .clist_set:N = \l_@@_borders_clist ,
6052     borders .value_required:n = true ,
6053     hvlines .meta:n = { vlines , hlines } ,
6054     vlines .bool_set:N = \l_@@_vlines_block_bool ,
6055     vlines .default:n = true ,
6056     hlines .bool_set:N = \l_@@_hlines_block_bool ,
6057     hlines .default:n = true ,
6058     line-width .dim_set:N = \l_@@_line_width_dim ,
6059     line-width .value_required:n = true ,
6060     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6061     l .value_forbidden:n = true ,
6062     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6063     r .value_forbidden:n = true ,
6064     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6065     c .value_forbidden:n = true ,
6066     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
6067         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6068     L .value_forbidden:n = true ,
6069     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6070         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6071     R .value_forbidden:n = true ,
6072     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6073         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6074     C .value_forbidden:n = true ,
6075     t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
6076     t .value_forbidden:n = true ,
6077     b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
6078     b .value_forbidden:n = true ,
6079     name .tl_set:N = \l_@@_block_name_str ,
6080     name .value_required:n = true ,
6081     name .initial:n = ,
6082     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,

```

```

6083   respect-arraystretch .default:n = true ,
6084   v-center .bool_set:N = \l_@@_v_center_bool ,
6085   v-center .default:n = true ,
6086   v-center .initial:n = false ,
6087   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
6088 }

```

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.

```

6089 \cs_new_protected:Npn \@@_draw_blocks:
6090 {
6091   \cs_set_eq:NN \ialign \@@_old_ialign:
6092   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
6093 }
6094 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
6095 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6096   \int_zero_new:N \l_@@_last_row_int
6097   \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.

```

6098   \int_compare:nNnTF { #3 } > { 99 }
6099     { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
6100       { \int_set:Nn \l_@@_last_row_int { #3 } }
6101   \int_compare:nNnTF { #4 } > { 99 }
6102     { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
6103       { \int_set:Nn \l_@@_last_col_int { #4 } }
6104   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6105     {
6106       \int_compare:nTF
6107         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6108         {
6109           \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
6110           \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
6111           \group_begin:
6112           \globaldefs = 1
6113           \@@_msg_redirect_name:nn { columns~not~used } { none }
6114           \group_end:
6115         }
6116         { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
6117     }
6118     {
6119       \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6120         { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
6121         { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6122     }
6123 }
6124 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6125 {

```

The group is for the keys.

```

6126   \group_begin:
6127   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }

```

We restrict the use of the key `v-center` to the case of a mono-row block.

```

6128 \bool_if:NT \l_@@_v_center_bool
6129 {
6130   \int_compare:nNnF { #1 } = { #3 }
6131   {
6132     \@@_error:n { Wrong~use~of~v-center }
6133     \bool_set_false:N \l_@@_v_center_bool
6134   }
6135 }
6136 \bool_if:NT \l_@@_vlines_block_bool
6137 {
6138   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6139   {
6140     \@@_vlines_block:nnn
6141     { \exp_not:n { #5 } }
6142     { #1 - #2 }
6143     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6144   }
6145 }
6146 \bool_if:NT \l_@@_hlines_block_bool
6147 {
6148   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6149   {
6150     \@@_hlines_block:nnn
6151     { \exp_not:n { #5 } }
6152     { #1 - #2 }
6153     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6154   }
6155 }
6156 \bool_if:nT
6157 { ! \l_@@_vlines_block_bool && ! \l_@@_hlines_block_bool }
6158 {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

6159 \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6160 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6161 }
6162 \tl_if_empty:NF \l_@@_draw_tl
6163 {
6164   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6165   {
6166     \@@_stroke_block:nnn
6167     { \exp_not:n { #5 } }
6168     { #1 - #2 }
6169     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6170   }
6171   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6172   { { #1 } { #2 } { #3 } { #4 } }
6173 }
6174 \clist_if_empty:NF \l_@@_borders_clist
6175 {
6176   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6177   {
6178     \@@_stroke_borders_block:nnn
6179     { \exp_not:n { #5 } }
6180     { #1 - #2 }
6181     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6182   }
6183 }
6184 \tl_if_empty:NF \l_@@_fill_tl
6185 {

```


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`.

```

6186     \exp_last_unbraced:N \@@_extract_brackets \l_@@_fill_tl \q_stop
6187     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6188     {
6189         \exp_not:N \roundedrectanglecolor
6190         [ \l_tmpa_tl ]
6191         { \exp_not:N \l_tmpb_tl }
6192         { #1 - #2 }
6193         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6194         { \dim_use:N \l_@@_rounded_corners_dim }
6195     }
6196 }
6197 \seq_if_empty:NF \l_@@_tikz_seq
6198 {
6199     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6200     {
6201         \@@_block_tikz:nnnnn
6202         { #1 }
6203         { #2 }
6204         { \int_use:N \l_@@_last_row_int }
6205         { \int_use:N \l_@@_last_col_int }
6206         { \seq_use:Nn \l_@@_tikz_seq { , } }
6207     }
6208 }

6209 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6210 {
6211     \tl_gput_right:Nx \g_@@_internal_code_after_tl
6212     {
6213         \@@_actually_diagbox:nnnnnn
6214         { #1 }
6215         { #2 }
6216         { \int_use:N \l_@@_last_row_int }
6217         { \int_use:N \l_@@_last_col_int }
6218         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6219     }
6220 }

6221 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6222 \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.

```

6223 \pgfpicture
6224 \pgfrememberpicturepositiononpagetrue
6225 \pgf@relevantforpicturesizefalse
6226 \@@_qpoint:n { row - #1 }
6227 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6228 \@@_qpoint:n { col - #2 }
6229 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6230 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6231 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6232 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6233 \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.

```

6234 \@@_pgf_rect_node:nnnnn
6235 { \@@_env: - #1 - #2 - block }
6236 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6237 \str_if_empty:NF \l_@@_block_name_str
6238 {
6239 \pgfnodealias
6240 { \@@_env: - \l_@@_block_name_str }
6241 { \@@_env: - #1 - #2 - block }
6242 \str_if_empty:NF \l_@@_name_str
6243 {
6244 \pgfnodealias
6245 { \l_@@_name_str - \l_@@_block_name_str }
6246 { \@@_env: - #1 - #2 - block }
6247 }
6248 }

```

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.

```

6249 \bool_if:NF \l_@@_hpos_of_block_cap_bool
6250 {
6251 \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.

```

6252 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6253 {

```

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.

```

6254 \cs_if_exist:cT
6255 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6256 {
6257 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6258 {
6259 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
6260 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
6261 }
6262 }
6263 }

```

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.

```

6264 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
6265 {
6266 \@@_qpoint:n { col - #2 }
6267 \dim_set_eq:NN \l_tmpb_dim \pgf@x

```

```

6268     }
6269     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
6270     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6271     {
6272         \cs_if_exist:cT
6273         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6274         {
6275             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6276             {
6277                 \pgfpointanchor
6278                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6279                 { east }
6280                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
6281             }
6282         }
6283     }
6284     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
6285     {
6286         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6287         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6288     }
6289     \@@_pgf_rect_node:nnnnn
6290     { \@@_env: - #1 - #2 - block - short }
6291     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6292 }

```

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.

```

6293     \bool_if:NT \l_@@_medium_nodes_bool
6294     {
6295         \@@_pgf_rect_node:nnn
6296         { \@@_env: - #1 - #2 - block - medium }
6297         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
6298         {
6299             \pgfpointanchor
6300             { \@@_env:
6301                 - \int_use:N \l_@@_last_row_int
6302                 - \int_use:N \l_@@_last_col_int - medium
6303             }
6304             { south-east }
6305         }
6306     }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

6307     \bool_if:nTF
6308     { \int_compare_p:nNn { #1 } = { #3 } && ! \l_@@_v_center_bool }
6309     {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

6310         \int_compare:nNnTF { #1 } = 0
6311         { \l_@@_code_for_first_row_tl }
6312         {
6313             \int_compare:nNnT { #1 } = \l_@@_last_row_int
6314             \l_@@_code_for_last_row_tl
6315         }

```

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`.

```

6316         \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

6317         \pgfpointanchor
6318         {

```

```

6319         \@@_env: - #1 - #2 - block
6320         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6321     }
6322     {
6323         \str_case:Vn \l_@@_hpos_block_str
6324         {
6325             c { center }
6326             l { west }
6327             r { east }
6328         }
6329     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

6330     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
6331     \pgfset { inner~sep = \c_zero_dim }
6332     \pgfnode
6333     { rectangle }
6334     {
6335         \str_case:Vn \l_@@_hpos_block_str
6336         {
6337             c { base }
6338             l { base~west }
6339             r { base~east }
6340         }
6341     }
6342     { \box_use_drop:N \l_@@_cell_box } { } { }
6343 }

```

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`).

```

6344     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

6345         \int_compare:nNnT { #2 } = 0
6346         { \str_set:Nn \l_@@_hpos_block_str r }
6347         \bool_if:nT \g_@@_last_col_found_bool
6348         {
6349             \int_compare:nNnT { #2 } = \g_@@_col_total_int
6350             { \str_set:Nn \l_@@_hpos_block_str l }
6351         }
6352         \pgftransformshift
6353         {
6354             \pgfpointanchor
6355             {
6356                 \@@_env: - #1 - #2 - block
6357                 \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6358             }
6359             {
6360                 \str_case:Vn \l_@@_hpos_block_str
6361                 {
6362                     c { center }
6363                     l { west }
6364                     r { east }
6365                 }
6366             }
6367         }
6368         \pgfset { inner~sep = \c_zero_dim }
6369         \pgfnode
6370         { rectangle }
6371         {
6372             \str_case:Vn \l_@@_hpos_block_str
6373             {
6374                 c { center }
6375                 l { west }

```

```

6376         r { east }
6377     }
6378 }
6379 { \box_use_drop:N \l_@@_cell_box } { } { }
6380 }
6381 \endpgfpicture
6382 \group_end:
6383 }

6384 \NewDocumentCommand \@@_extract_brackets { 0 { } }
6385 {
6386     \tl_set:Nn \l_tmpa_tl { #1 }
6387     \@@_store_in_tmpb_tl
6388 }
6389 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
6390 { \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).

```

6391 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
6392 {
6393     \group_begin:
6394     \tl_clear:N \l_@@_draw_tl
6395     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6396     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
6397     \pgfpicture
6398     \pgfrememberpicturerepositiononpagetrue
6399     \pgf@relevantforpicturesizefalse
6400     \tl_if_empty:NF \l_@@_draw_tl
6401     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

6402         \str_if_eq:VnTF \l_@@_draw_tl { default }
6403         { \CT@arc@ }
6404         { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
6405     }
6406     \pgfsetcornersarced
6407     {
6408         \pgfpoint
6409         { \dim_use:N \l_@@_rounded_corners_dim }
6410         { \dim_use:N \l_@@_rounded_corners_dim }
6411     }
6412     \@@_cut_on_hyphen:w #2 \q_stop
6413     \bool_lazy_and:nnT
6414     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
6415     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
6416     {
6417         \@@_qpoint:n { row - \l_tmpa_tl }
6418         \dim_set:Nn \l_tmpb_dim { \pgf@y }
6419         \@@_qpoint:n { col - \l_tmpb_tl }
6420         \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
6421         \@@_cut_on_hyphen:w #3 \q_stop
6422         \int_compare:nNnT \l_tmpa_tl > \c@iRow
6423         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
6424         \int_compare:nNnT \l_tmpb_tl > \c@jCol
6425         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
6426         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
6427         \dim_set:Nn \l_tmpa_dim { \pgf@y }
6428         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6429         \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }

```

```

6430     \pgfpathrectanglecorners
6431     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6432     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6433     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

6434     \pgfusepath { stroke }
6435 }
6436 \endpgfpicture
6437 \group_end:
6438 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

6439 \keys_define:nn { NiceMatrix / BlockStroke }
6440 {
6441   color .tl_set:N = \l_@@_draw_tl ,
6442   draw .tl_set:N = \l_@@_draw_tl ,
6443   draw .default:n = default ,
6444   line-width .dim_set:N = \l_@@_line_width_dim ,
6445   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6446   rounded-corners .default:n = 4 pt
6447 }

```

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

6448 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
6449 {
6450   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6451   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6452   \@@_cut_on_hyphen:w #2 \q_stop
6453   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6454   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6455   \@@_cut_on_hyphen:w #3 \q_stop
6456   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6457   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6458   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
6459   {
6460     \use:x
6461     {
6462       \@@_vline:n
6463       {
6464         position = ##1 ,
6465         start = \l_@@_tmpc_tl ,
6466         end = \int_eval:n { \l_tmpa_tl - 1 }
6467       }
6468     }
6469   }
6470 }
6471 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
6472 {
6473   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6474   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6475   \@@_cut_on_hyphen:w #2 \q_stop
6476   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6477   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6478   \@@_cut_on_hyphen:w #3 \q_stop
6479   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6480   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6481   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
6482   {
6483     \use:x
6484     {
6485       \@@_hline:n

```

```

6486         {
6487             position = ##1 ,
6488             start = \l_@@_tmpd_tl ,
6489             end = \int_eval:n { \l_tmpb_tl - 1 } ,
6490             total-width = \arrayrulewidth
6491         }
6492     }
6493 }
6494 }

```

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).

```

6495 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
6496 {
6497     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6498     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6499     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
6500     { \@@_error:n { borders~forbidden } }
6501     {
6502         \tl_clear_new:N \l_@@_borders_tikz_tl
6503         \keys_set:nV
6504             { NiceMatrix / OnlyForTikzInBorders }
6505         \l_@@_borders_clist
6506         \@@_cut_on_hyphen:w #2 \q_stop
6507         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6508         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6509         \@@_cut_on_hyphen:w #3 \q_stop
6510         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6511         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6512         \@@_stroke_borders_block_i:
6513     }
6514 }
6515 \hook_gput_code:nnn { begindocument } { . }
6516 {
6517     \cs_new_protected:Npx \@@_stroke_borders_block_i:
6518     {
6519         \c_@@_pgfortikzpicture_tl
6520         \@@_stroke_borders_block_ii:
6521         \c_@@_endpgfortikzpicture_tl
6522     }
6523 }
6524 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
6525 {
6526     \pgfrememberpicturepositiononpagetrue
6527     \pgf@relevantforpicturesizefalse
6528     \CT@arc@
6529     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
6530     \clist_if_in:NnT \l_@@_borders_clist { right }
6531     { \@@_stroke_vertical:n \l_tmpb_tl }
6532     \clist_if_in:NnT \l_@@_borders_clist { left }
6533     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
6534     \clist_if_in:NnT \l_@@_borders_clist { bottom }
6535     { \@@_stroke_horizontal:n \l_tmpa_tl }
6536     \clist_if_in:NnT \l_@@_borders_clist { top }
6537     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
6538 }
6539 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
6540 {
6541     tikz .code:n =
6542         \cs_if_exist:NTF \tikzpicture
6543         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }

```

```

6544     { \@@_error:n { tikz-in-borders-without-tikz } } ,
6545     tikz .value_required:n = true ,
6546     top .code:n = ,
6547     bottom .code:n = ,
6548     left .code:n = ,
6549     right .code:n = ,
6550     unknown .code:n = \@@_error:n { bad-border }
6551 }

```

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).

```

6552 \cs_new_protected:Npn \@@_stroke_vertical:n #1
6553 {
6554   \@@_qpoint:n \l_@@_tmpc_tl
6555   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6556   \@@_qpoint:n \l_tmpa_tl
6557   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6558   \@@_qpoint:n { #1 }
6559   \tl_if_empty:NTF \l_@@_borders_tikz_tl
6560   {
6561     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
6562     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
6563     \pgfusepathqstroke
6564   }
6565   {
6566     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
6567     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
6568   }
6569 }

```

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).

```

6570 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
6571 {
6572   \@@_qpoint:n \l_@@_tmpd_tl
6573   \clist_if_in:NnTF \l_@@_borders_clist { left }
6574   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
6575   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
6576   \@@_qpoint:n \l_tmpb_tl
6577   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
6578   \@@_qpoint:n { #1 }
6579   \tl_if_empty:NTF \l_@@_borders_tikz_tl
6580   {
6581     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
6582     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6583     \pgfusepathqstroke
6584   }
6585   {
6586     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
6587     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
6588   }
6589 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

6590 \keys_define:nn { NiceMatrix / BlockBorders }
6591 {
6592   borders .clist_set:N = \l_@@_borders_clist ,
6593   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6594   rounded-corners .default:n = 4 pt ,
6595   line-width .dim_set:N = \l_@@_line_width_dim ,
6596 }

```


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.

```

6597 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
6598 {
6599   \begin { tikzpicture }
6600   \clist_map_inline:nn { #5 }
6601     {
6602       \path [ ##1 ]
6603         ( #1 -| #2 )
6604         rectangle
6605         ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
6606     }
6607   \end { tikzpicture }
6608 }

```

How to draw the dotted lines transparently

```

6609 \cs_set_protected:Npn \@@_renew_matrix:
6610 {
6611   \RenewDocumentEnvironment { pmatrix } { } {
6612     { \pNiceMatrix }
6613     { \endpNiceMatrix }
6614   \RenewDocumentEnvironment { vmatrix } { } {
6615     { \vNiceMatrix }
6616     { \endvNiceMatrix }
6617   \RenewDocumentEnvironment { Vmatrix } { } {
6618     { \VNiceMatrix }
6619     { \endVNiceMatrix }
6620   \RenewDocumentEnvironment { bmatrix } { } {
6621     { \bNiceMatrix }
6622     { \endbNiceMatrix }
6623   \RenewDocumentEnvironment { Bmatrix } { } {
6624     { \BNiceMatrix }
6625     { \endBNiceMatrix }
6626 }

```

Automatic arrays

```

6627 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
6628 {
6629   \int_set:Nn \l_@@_nb_rows_int { #1 }
6630   \int_set:Nn \l_@@_nb_cols_int { #2 }
6631 }

```

We will extract the potential keys `l`, `r` and `c` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

6632 \keys_define:nn { NiceMatrix / Auto }
6633 {
6634   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
6635   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
6636   c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
6637 }
6638 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
6639 {
6640   \int_zero_new:N \l_@@_nb_rows_int
6641   \int_zero_new:N \l_@@_nb_cols_int
6642   \@@_set_size:n #4 \q_stop

```

The group is for the protection of `\l_@@_type_of_col_tl`.

```

6643   \group_begin:

```

```

6644 \tl_set:Nn \l_@@_type_of_col_tl c
6645 \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
6646 \use:x
6647 {
6648   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
6649   { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
6650   [ \exp_not:V \l_tmpa_tl ]
6651 }
6652 \int_compare:nNnT \l_@@_first_row_int = 0
6653 {
6654   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6655   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6656   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6657 }
6658 \prg_replicate:nn \l_@@_nb_rows_int
6659 {
6660   \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).

```

6661   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
6662   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6663 }
6664 \int_compare:nNnT \l_@@_last_row_int > { -2 }
6665 {
6666   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6667   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6668   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6669 }
6670 \end { NiceArrayWithDelims }
6671 \group_end:
6672 }
6673 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6674 {
6675   \cs_set_protected:cpn { #1 AutoNiceMatrix }
6676   {
6677     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6678     \AutoNiceMatrixWithDelims { #2 } { #3 }
6679   }
6680 }
6681 \@@_define_com:nnn p ( )
6682 \@@_define_com:nnn b [ ]
6683 \@@_define_com:nnn v | |
6684 \@@_define_com:nnn V \ | \ |
6685 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

6686 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
6687 {
6688   \group_begin:
6689   \bool_set_true:N \l_@@_NiceArray_bool
6690   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6691   \group_end:
6692 }

```

The redefinition of the command \dotfill

```

6693 \cs_set_eq:NN \@@_old_dotfill \dotfill
6694 \cs_new_protected:Npn \@@_dotfill:
6695 {

```

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}`).

```

6696 \@@_old_dotfill
6697 \bool_if:NT \l_@@_NiceTabular_bool
6698 { \group_insert_after:N \@@_dotfill_ii: }
6699 { \group_insert_after:N \@@_dotfill_i: }
6700 }
6701 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
6702 \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`.

```

6703 \cs_new_protected:Npn \@@_dotfill_iii:
6704 { \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.

```

6705 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6706 {
6707   \tl_gput_right:Nx \g_@@_internal_code_after_tl
6708   {
6709     \@@_actually_diagbox:nnnnnn
6710     { \int_use:N \c@iRow }
6711     { \int_use:N \c@jCol }
6712     { \int_use:N \c@iRow }
6713     { \int_use:N \c@jCol }
6714     { \exp_not:n { #1 } }
6715     { \exp_not:n { #2 } }
6716   }

```

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`.

```

6717 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6718 {
6719   { \int_use:N \c@iRow }
6720   { \int_use:N \c@jCol }
6721   { \int_use:N \c@iRow }
6722   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

6723   { }
6724 }
6725 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it’s possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

6726 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
6727 {
6728   \pgfpicture
6729   \pgf@relevantforpicturesizefalse
6730   \pgfrememberpicturepositiononpagetrue
6731   \@@_qpoint:n { row - #1 }
6732   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6733   \@@_qpoint:n { col - #2 }
6734   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6735   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6736   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
6737   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y

```

```

6738 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
6739 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6740 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6741 {

```

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.

```

6742 \CT@arc@
6743 \pgfsetroundcap
6744 \pgfusepathqstroke
6745 }
6746 \pgfset { inner~sep = 1 pt }
6747 \pgfscope
6748 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6749 \pgfnode { rectangle } { south-west }
6750 {
6751 \begin { minipage } { 20 cm }
6752 \@@_math_toggle_token: #5 \@@_math_toggle_token:
6753 \end { minipage }
6754 }
6755 { }
6756 { }
6757 \endpgfscope
6758 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6759 \pgfnode { rectangle } { north-east }
6760 {
6761 \begin { minipage } { 20 cm }
6762 \raggedleft
6763 \@@_math_toggle_token: #6 \@@_math_toggle_token:
6764 \end { minipage }
6765 }
6766 { }
6767 { }
6768 \endpgfpicture
6769 }

```

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.

```

6770 \keys_define:nn { NiceMatrix }
6771 {
6772 CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6773 CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6774 }
6775 \keys_define:nn { NiceMatrix / CodeAfter }
6776 {
6777 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6778 sub-matrix .value_required:n = true ,
6779 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6780 delimiters / color .value_required:n = true ,
6781 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6782 rules .value_required:n = true ,
6783 unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
6784 }

```

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. 124.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
6785 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_ii:n }
```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_ii:n` which begins with `\`.

```
6786 \cs_new_protected:Npn \@@_CodeAfter_i: { \ \omit \@@_CodeAfter_ii:n }
```

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```
6787 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
6788 {
6789   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6790   \@@_CodeAfter_iv:n
6791 }
```

We catch the argument of the command `\end` (in `#1`).

```
6792 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
6793 {
```

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.

```
6794   \str_if_eq:eeTF \currentenv { #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`.

```
6795   {
6796     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6797     \@@_CodeAfter_ii:n
6798   }
6799 }
```

The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_internal_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
6800 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
6801 {
6802   \pgfpicture
6803   \pgfrememberpicturepositiononpagetrue
6804   \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.

```
6805   \@@_qpoint:n { row - 1 }
6806   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6807   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
6808   \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).

```
6809   \bool_if:nTF { #3 }
6810     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6811     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6812   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
```

```

6813 {
6814   \cs_if_exist:cT
6815   { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
6816   {
6817     \pgfpointanchor
6818     { \l_@@_env: - ##1 - #2 }
6819     { \bool_if:nTF { #3 } { west } { east } }
6820     \dim_set:Nn \l_tmpa_dim
6821     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6822   }
6823 }

```

Now we can put the delimiter with a node of PGF.

```

6824 \pgfset { inner~sep = \c_zero_dim }
6825 \dim_zero:N \nulldelimiterspace
6826 \pgftransformshift
6827 {
6828   \pgfpoint
6829   { \l_tmpa_dim }
6830   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6831 }
6832 \pgfnode
6833 { rectangle }
6834 { \bool_if:nTF { #3 } { east } { west } }
6835 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

6836   \nullfont
6837   \c_math_toggle_token
6838   \tl_if_empty:NF \l_@@_delimiters_color_tl
6839   { \color { \l_@@_delimiters_color_tl } }
6840   \bool_if:nTF { #3 } { \left #1 } { \left . }
6841   \vcenter
6842   {
6843     \nullfont
6844     \hrule \@height
6845       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6846       \@depth \c_zero_dim
6847       \@width \c_zero_dim
6848   }
6849   \bool_if:nTF { #3 } { \right . } { \right #1 }
6850   \c_math_toggle_token
6851 }
6852 { }
6853 { }
6854 \endpgfpicture
6855 }

```

The command `\SubMatrix`

```

6856 \keys_define:nn { NiceMatrix / sub-matrix }
6857 {
6858   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6859   extra-height .value_required:n = true ,
6860   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6861   left-xshift .value_required:n = true ,
6862   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6863   right-xshift .value_required:n = true ,
6864   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6865   xshift .value_required:n = true ,
6866   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6867   delimiters / color .value_required:n = true ,
6868   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6869   slim .default:n = true ,

```

```

6870 hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6871 hlines .default:n = all ,
6872 vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6873 vlines .default:n = all ,
6874 hvlines .meta:n = { hlines, vlines } ,
6875 hvlines .value_forbidden:n = true ,
6876 }
6877 \keys_define:nn { NiceMatrix }
6878 {
6879   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6880   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6881   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6882   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6883   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6884   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6885 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

6886 \keys_define:nn { NiceMatrix / SubMatrix }
6887 {
6888   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6889   delimiters / color .value_required:n = true ,
6890   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6891   hlines .default:n = all ,
6892   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6893   vlines .default:n = all ,
6894   hvlines .meta:n = { hlines, vlines } ,
6895   hvlines .value_forbidden:n = true ,
6896   name .code:n =
6897     \tl_if_empty:nTF { #1 }
6898     { \@@_error:n { Invalid-name } }
6899     {
6900       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6901       {
6902         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6903         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
6904         {
6905           \str_set:Nn \l_@@_submatrix_name_str { #1 }
6906           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6907         }
6908       }
6909       { \@@_error:n { Invalid-name } }
6910     } ,
6911   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6912   rules .value_required:n = true ,
6913   code .tl_set:N = \l_@@_code_tl ,
6914   code .value_required:n = true ,
6915   name .value_required:n = true ,
6916   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
6917 }

6918 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
6919 {
6920   \peek_remove_spaces:n
6921   {
6922     \@@_cut_on_hyphen:w #3 \q_stop
6923     \tl_clear_new:N \l_@@_tmpc_tl
6924     \tl_clear_new:N \l_@@_tmpd_tl
6925     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6926     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6927     \@@_cut_on_hyphen:w #2 \q_stop
6928     \seq_gput_right:Nx \g_@@_submatrix_seq

```

```

6929      { { \l_tmpa_tl } { \l_tmpb_tl } { \l_@@_tmpc_tl } { \l_@@_tmpd_tl } }
6930      \tl_gput_right:Nn \g_@@_internal_code_after_tl
6931      { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6932    }
6933  }

```

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.

```

6934 \hook_gput_code:nnn { begindocument } { . }
6935 {
6936   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
6937   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
6938   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
6939     {
6940       \peek_remove_spaces:n
6941       {
6942         \@@_sub_matrix:nnnnnnn
6943         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
6944       }
6945     }
6946 }

```

The following macro will compute \l_@@_first_i_tl, \l_@@_first_j_tl, \l_@@_last_i_tl and \l_@@_last_j_tl from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

6947 \cs_new_protected:Npn \@@_compute_i_j:nn #1 #2
6948 {
6949   \tl_clear_new:N \l_@@_first_i_tl
6950   \tl_clear_new:N \l_@@_first_j_tl
6951   \tl_clear_new:N \l_@@_last_i_tl
6952   \tl_clear_new:N \l_@@_last_j_tl
6953   \@@_cut_on_hyphen:w #1 \q_stop
6954   \tl_if_eq:NnTF \l_tmpa_tl { last }
6955     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
6956     { \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl }
6957   \tl_if_eq:NnTF \l_tmpb_tl { last }
6958     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
6959     { \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl }
6960   \@@_cut_on_hyphen:w #2 \q_stop
6961   \tl_if_eq:NnTF \l_tmpa_tl { last }
6962     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
6963     { \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl }
6964   \tl_if_eq:NnTF \l_tmpb_tl { last }
6965     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
6966     { \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl }
6967 }

```



```

6968 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6969 {
6970   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

6971   \@@_compute_i_j:nn { #2 } { #3 }
6972   \bool_lazy_or:nnTF
6973     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6974     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6975     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
6976     {
6977       \str_clear_new:N \l_@@_submatrix_name_str
6978       \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6979       \pgfpicture
6980       \pgfrememberpicturepositiononpagetrue
6981       \pgf@relevantforpicturesizefalse
6982       \pgfset { inner~sep = \c_zero_dim }
6983       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6984       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currification.

```

6985       \bool_if:NTF \l_@@_submatrix_slim_bool
6986       { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6987       { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6988       {
6989         \cs_if_exist:cT
6990           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6991           {
6992             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6993             \dim_set:Nn \l_@@_x_initial_dim
6994               { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6995           }
6996         \cs_if_exist:cT
6997           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6998           {
6999             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7000             \dim_set:Nn \l_@@_x_final_dim
7001               { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7002           }
7003       }
7004       \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
7005       { \@@_error:nn { Impossible-delimiter } { left } }
7006       {
7007         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
7008         { \@@_error:nn { Impossible-delimiter } { right } }
7009         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
7010       }
7011       \endpgfpicture
7012     }
7013   \group_end:
7014 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

7015 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
7016 {
7017   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
7018   \dim_set:Nn \l_@@_y_initial_dim
7019     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
7020   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
7021   \dim_set:Nn \l_@@_y_final_dim
7022     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
7023   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
7024     {
7025       \cs_if_exist:cT

```

```

7026 { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
7027 {
7028   \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
7029   \dim_set:Nn \l_@@_y_initial_dim
7030     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
7031 }
7032 \cs_if_exist:cT
7033 { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
7034 {
7035   \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
7036   \dim_set:Nn \l_@@_y_final_dim
7037     { \dim_min:nn \l_@@_y_final_dim \pgf@y }
7038 }
7039 }
7040 \dim_set:Nn \l_tmpa_dim
7041 {
7042   \l_@@_y_initial_dim - \l_@@_y_final_dim +
7043   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
7044 }
7045 \dim_zero:N \nullldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

7046 \group_begin:
7047 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7048 \tl_if_empty:NF \l_@@_rules_color_tl
7049 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
7050 \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`.

```

7051 \seq_map_inline:Nn \g_@@_cols_vlism_seq
7052 {
7053   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
7054   {
7055     \int_compare:nNnT
7056       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
7057     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

7058       \@@_qpoint:n { col - ##1 }
7059       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7060       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7061       \pgfusepathqstroke
7062     }
7063   }
7064 }

```

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.

```

7065 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
7066 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
7067 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
7068 {
7069   \bool_lazy_and:nnTF
7070     { \int_compare_p:nNn { ##1 } > 0 }
7071     {
7072       \int_compare_p:nNn
7073         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7074     {
7075       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7076       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7077       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }

```

```

7078         \pgfusepathqstroke
7079     }
7080     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
7081 }

```

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.

```

7082     \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7083     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7084     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
7085     {
7086         \bool_lazy_and:nnTF
7087         { \int_compare_p:nNn { ##1 } > 0 }
7088         {
7089             \int_compare_p:nNn
7090             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7091         {
7092             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7093         \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
7094         \dim_set:Nn \l_tmpa_dim
7095         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7096         \str_case:nn { #1 }
7097         {
7098             ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7099             [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7100             \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7101         }
7102         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

7103         \dim_set:Nn \l_tmpb_dim
7104         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7105         \str_case:nn { #2 }
7106         {
7107             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7108             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7109             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7110         }
7111         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7112         \pgfusepathqstroke
7113         \group_end:
7114     }
7115     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7116 }

```

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).

```

7117     \str_if_empty:NF \l_@@_submatrix_name_str
7118     {
7119         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
7120         \l_@@_x_initial_dim \l_@@_y_initial_dim
7121         \l_@@_x_final_dim \l_@@_y_final_dim
7122     }
7123     \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`.

```

7124     \begin { pgfscope }
7125     \pgftransformshift

```

```

7126 {
7127   \pgfpoint
7128   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7129   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7130 }
7131 \str_if_empty:NTF \l_@@_submatrix_name_str
7132 { \@@_node_left:nn #1 { } }
7133 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7134 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7135 \pgftransformshift
7136 {
7137   \pgfpoint
7138   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7139   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7140 }
7141 \str_if_empty:NTF \l_@@_submatrix_name_str
7142 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7143 {
7144   \@@_node_right:nnnn #2
7145   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7146 }
7147 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7148 \flag_clear_new:n { nicematrix }
7149 \l_@@_code_tl
7150 }

```

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`.

```

7151 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpntanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

7152 \cs_new_protected:Npn \@@_pgfpntanchor:n #1
7153 {
7154   \use:e
7155   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpntanchor_i:nn #1 } }
7156 }

```

In fact, the argument of `\pgfpointanchor` is always of the form `\a_command { name_of_node }` where “name_of_node” is the name of the Tikz node without the potential prefix and suffix. That's why we catch two arguments and work only on the second by trying (first) to extract an hyphen -.

```

7157 \cs_new:Npn \@@_pgfpntanchor_i:nn #1 #2
7158 { #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.

```

7159 \tl_const:Nn \c_@@_integers_alist_tl
7160 {
7161   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7162   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7163   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7164   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7165 }

```

```

7166 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
7167 {

```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form $i-j$. In that case, the i of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the j arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

7168   \tl_if_empty:nTF { #2 }
7169   {
7170     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7171     {
7172       \flag_raise:n { nicematrix }
7173       \int_if_even:nTF { \flag_height:n { nicematrix } }
7174       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7175       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7176     }
7177     { #1 }
7178   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

7179   { \@@_pgfpointanchor_iii:w { #1 } #2 }
7180 }

```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

7181 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
7182 {
7183   \str_case:nnF { #1 }
7184   {
7185     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7186     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7187   }

```

Now the case of a node of the form $i-j$.

```

7188   {
7189     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7190     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7191   }
7192 }

```

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`).

```

7193 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7194 {
7195   \pgfnode
7196   { rectangle }
7197   { east }
7198   {
7199     \nullfont
7200     \c_math_toggle_token
7201     \tl_if_empty:NF \l_@@_delimiters_color_tl
7202     { \color { \l_@@_delimiters_color_tl } }
7203     \left #1
7204     \vcenter
7205     {
7206       \nullfont
7207       \hrule \@height \l_tmpa_dim
7208       \@depth \c_zero_dim
7209       \@width \c_zero_dim
7210     }

```

```

7211     \right .
7212     \c_math_toggle_token
7213   }
7214   { #2 }
7215   { }
7216 }

```

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.

```

7217 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
7218 {
7219   \pgfnode
7220   { rectangle }
7221   { west }
7222   {
7223     \nullfont
7224     \c_math_toggle_token
7225     \tl_if_empty:NF \l_@@_delimiters_color_tl
7226     { \color { \l_@@_delimiters_color_tl } }
7227     \left .
7228     \vcenter
7229     {
7230       \nullfont
7231       \hrule \@height \l_tmpa_dim
7232             \@depth \c_zero_dim
7233             \@width \c_zero_dim
7234     }
7235     \right #1
7236     \tl_if_empty:NF { #3 } { _ { \smash { #3 } } }
7237     ^ { \smash { #4 } }
7238     \c_math_toggle_token
7239   }
7240   { #2 }
7241   { }
7242 }

```

Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

7243 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
7244 {
7245   \peek_remove_spaces:n
7246   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
7247 }
7248 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
7249 {
7250   \peek_remove_spaces:n
7251   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
7252 }
7253 \keys_define:nn { NiceMatrix / Brace }
7254 {
7255   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
7256   left-shorten .default:n = true ,
7257   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
7258   shorten .meta:n = { left-shorten , right-shorten } ,
7259   right-shorten .default:n = true ,
7260   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
7261   yshift .value_required:n = true ,

```

```

7262 yshift .initial:n = \c_zero_dim ,
7263 color .tl_set:N = \l_tmpa_tl ,
7264 color .value_required:n = true ,
7265 unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
7266 }

```

#1 is the first cell of the rectangle (with the syntax $i-j$; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of key-value pairs); #5 is equal to under or over.

```

7267 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
7268 {
7269   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

7270   \@@_compute_i_j:nn { #1 } { #2 }
7271   \bool_lazy_or:nnTF
7272     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7273     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7274     {
7275       \str_if_eq:nnTF { #5 } { under }
7276       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
7277       { \@@_error:nn { Construct~too~large } { \OverBrace } }
7278     }
7279   {
7280     \tl_clear:N \l_tmpa_tl % added the 2022-02-25
7281     \keys_set:nn { NiceMatrix / Brace } { #4 }
7282     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } } % added the 2022-02-25
7283     \pgfpicture
7284     \pgfrememberpicturepositiononpagetrue
7285     \pgf@relevantforpicturesizefalse
7286     \bool_if:NT \l_@@_brace_left_shorten_bool
7287     {
7288       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7289       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7290       {
7291         \cs_if_exist:cT
7292           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7293           {
7294             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7295             \dim_set:Nn \l_@@_x_initial_dim
7296               { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7297           }
7298       }
7299     }
7300     \bool_lazy_or:nnT
7301       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
7302       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
7303       {
7304         \@@_qpoint:n { col - \l_@@_first_j_tl }
7305         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7306       }
7307     \bool_if:NT \l_@@_brace_right_shorten_bool
7308     {
7309       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
7310       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7311       {
7312         \cs_if_exist:cT
7313           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7314           {
7315             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7316             \dim_set:Nn \l_@@_x_final_dim
7317               { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7318           }
7319       }

```

```

7320     }
7321     \bool_lazy_or:nnT
7322     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
7323     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
7324     {
7325         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
7326         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7327     }
7328     \pgfset { inner~sep = \c_zero_dim }
7329     \str_if_eq:nnTF { #5 } { under }
7330     { \@@_underbrace_i:n { #3 } }
7331     { \@@_overbrace_i:n { #3 } }
7332     \endpgfpicture
7333 }
7334 \group_end:
7335 }

```

The argument is the text to put above the brace.

```

7336 \cs_new_protected:Npn \@@_overbrace_i:n #1
7337 {
7338     \@@_qpoint:n { row - \l_@@_first_i_tl }
7339     \pgftransformshift
7340     {
7341         \pgfpoint
7342         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7343         { \pgf@y + \l_@@_brace_yshift_dim }
7344     }
7345     \pgfnode
7346     { rectangle }
7347     { south }
7348     {
7349         \vbox_top:n
7350         {
7351             \group_begin:
7352             \everycr { }
7353             \halign
7354             {
7355                 \hfil ## \hfil \crrc
7356                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7357                 \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7358                 \hbox_to_wd:nn
7359                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7360                 { \downbracefill } \cr
7361             }
7362             \group_end:
7363         }
7364     }
7365     { }
7366     { }
7367 }

```

The argument is the text to put under the brace.

```

7368 \cs_new_protected:Npn \@@_underbrace_i:n #1
7369 {
7370     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
7371     \pgftransformshift
7372     {
7373         \pgfpoint
7374         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7375         { \pgf@y - \l_@@_brace_yshift_dim }
7376     }
7377     \pgfnode
7378     { rectangle }

```



```

7379 { north }
7380 {
7381   \group_begin:
7382   \everycr { }
7383   \vbox:n
7384   {
7385     \halign
7386     {
7387       \hfil ## \hfil \cr
7388       \hbox_to_wd:nn
7389       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7390       { \upbracefill } \cr
7391       \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7392       \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7393     }
7394   }
7395   \group_end:
7396 }
7397 { }
7398 { }
7399 }

```

The command \ShowCellNames

```

7400 \NewDocumentCommand \@@_ShowCellNames { }
7401 {
7402   \dim_zero_new:N \g_@@_tmpc_dim
7403   \dim_zero_new:N \g_@@_tmpd_dim
7404   \dim_zero_new:N \g_@@_tmpe_dim
7405   \int_step_inline:nn \c@iRow
7406   {
7407     \begin { pgfpicture }
7408     \@@_qpoint:n { row - ##1 }
7409     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7410     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
7411     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
7412     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
7413     \end { pgfpicture }
7414     \int_step_inline:nn \c@jCol
7415     {
7416       \hbox_set:Nn \l_tmpa_box
7417       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
7418       \begin { pgfpicture }
7419       \@@_qpoint:n { col - #####1 }
7420       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
7421       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
7422       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
7423       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
7424       \end { pgfpicture }
7425       \fp_set:Nn \l_tmpa_fp
7426       {
7427         \fp_min:nn
7428         {
7429           \fp_min:nn
7430           { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
7431           { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
7432         }
7433         { 1.0 }
7434       }
7435       \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
7436       \pgfpicture
7437       \pgfrememberpicturepositiononpagetrue

```

```

7438     \pgf@relevantforpicturesizefalse
7439     \pgftransformshift
7440     {
7441         \pgfpoint
7442         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
7443         { \dim_use:N \g_tmpa_dim }
7444     }
7445     \pgfnode
7446     { rectangle }
7447     { center }
7448     { \box_use:N \l_tmpa_box }
7449     { }
7450     { }
7451     \endpgfpicture
7452 }
7453 }
7454 }

```

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 `\c_@@_messages_for_Overleaf_bool` corresponds to the key `messages-for-Overleaf`.

```

7455 \bool_new:N \c_@@_messages_for_Overleaf_bool

```

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

7456 \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.

```

7457 \bool_new:N \c_@@_footnote_bool
7458 \msg_new:nnnn { nicematrix } { Unknown-key-for-package }
7459 {
7460     The~key~'\l_keys_key_str'~is~unknown. \\
7461     That~key~will~be~ignored. \\
7462     For~a~list~of~the~available~keys,~type~H~<return>.
7463 }
7464 {
7465     The~available~keys~are~(in~alphabetic~order):~
7466     footnote,~
7467     footnotehyper,~
7468     messages-for-Overleaf,~
7469     renew-dots,~and
7470     renew-matrix.
7471 }
7472 \keys_define:nn { NiceMatrix / Package }
7473 {
7474     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
7475     renew-dots .value_forbidden:n = true ,
7476     renew-matrix .code:n = \@@_renew_matrix: ,
7477     renew-matrix .value_forbidden:n = true ,
7478     messages-for-Overleaf .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
7479     footnote .bool_set:N = \c_@@_footnote_bool ,
7480     footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
7481     unknown .code:n = \@@_error:n { Unknown-key-for-package }
7482 }
7483 \ProcessKeysOptions { NiceMatrix / Package }

```

```

7484 \@@_msg_new:nn { footnote-with-footnotehyper-package }
7485 {
7486   You~can't~use~the~option~'footnote'~because~the~package~
7487   footnotehyper~has~already~been~loaded.~
7488   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
7489   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7490   of~the~package~footnotehyper.\\
7491   The~package~footnote~won't~be~loaded.
7492 }
7493 \@@_msg_new:nn { footnotehyper-with-footnote-package }
7494 {
7495   You~can't~use~the~option~'footnotehyper'~because~the~package~
7496   footnote~has~already~been~loaded.~
7497   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
7498   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7499   of~the~package~footnote.\\
7500   The~package~footnotehyper~won't~be~loaded.
7501 }

7502 \bool_if:NT \c_@@_footnote_bool
7503 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7504   \@ifclassloaded { beamer }
7505   { \bool_set_false:N \c_@@_footnote_bool }
7506   {
7507     \@ifpackageloaded { footnotehyper }
7508     { \@@_error:n { footnote-with-footnotehyper-package } }
7509     { \usepackage { footnote } }
7510   }
7511 }

7512 \bool_if:NT \c_@@_footnotehyper_bool
7513 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7514   \@ifclassloaded { beamer }
7515   { \bool_set_false:N \c_@@_footnote_bool }
7516   {
7517     \@ifpackageloaded { footnote }
7518     { \@@_error:n { footnotehyper-with-footnote-package } }
7519     { \usepackage { footnotehyper } }
7520   }
7521   \bool_set_true:N \c_@@_footnote_bool
7522 }

```

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

```

7523 \bool_if:NTF \c_@@_messages_for_Overleaf_bool
7524 { \str_const:Nn \c_@@_available_keys_str { } }
7525 {
7526   \str_const:Nn \c_@@_available_keys_str
7527   { For~a~list~of~the~available~keys,~type~H~<return>. }
7528 }

7529 \seq_new:N \g_@@_types_of_matrix_seq
7530 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
7531 {

```

```

7532     NiceMatrix ,
7533     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
7534 }
7535 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
7536 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NVTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

7537 \cs_new_protected:Npn \@@_error_too_much_cols:
7538 {
7539     \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
7540     {
7541         \int_compare:nNnTF \l_@@_last_col_int = { -2 }
7542         { \@@_fatal:n { too~much~cols~for~matrix } }
7543         {
7544             \bool_if:NF \l_@@_last_col_without_value_bool
7545             { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
7546         }
7547     }
7548     { \@@_fatal:n { too~much~cols~for~array } }
7549 }

```

The following command must *not* be protected since it's used in an error message.

```

7550 \cs_new:Npn \@@_message_hdotsfor:
7551 {
7552     \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
7553     { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
7554 }
7555 \@@_msg_new:nn { negative~weight }
7556 {
7557     Negative~weight.\\
7558     The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
7559     the~value~'#1'.\\
7560     The~absolute~value~will~be~used.
7561 }
7562 \@@_msg_new:nn { last~col~not~used }
7563 {
7564     Column~not~used.\\
7565     The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
7566     in~your~\@@_full_name_env:.~However,~you~can~go~on.
7567 }
7568 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
7569 {
7570     Too~much~columns.\\
7571     In~the~row~\int_eval:n { \c@jCol - 1 },~
7572     you~try~to~use~more~columns~
7573     than~allowed~by~your~\@@_full_name_env:.~\@@_message_hdotsfor:\
7574     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
7575     (plus~the~exterior~columns).~This~error~is~fatal.
7576 }
7577 \@@_msg_new:nn { too~much~cols~for~matrix }
7578 {
7579     Too~much~columns.\\
7580     In~the~row~\int_eval:n { \c@jCol - 1 },~
7581     you~try~to~use~more~columns~than~allowed~by~your~
7582     \@@_full_name_env:.~\@@_message_hdotsfor:\ Recall~that~the~maximal~
7583     number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
7584     'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
7585     This~error~is~fatal.
7586 }

```

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.

```

7587 \@@_msg_new:nn { too-much-cols-for-array }
7588 {
7589   Too-much-columns.\
7590   In-the-row~\int_eval:n { \c@jCol - 1 },~
7591   ~you-try-to-use-more-columns-than-allowed-by-your~
7592   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
7593   \int_use:N \g_@@_static_num_of_col_int\
7594   ~(plus-the-potential-exterior-ones).~
7595   This-error-is-fatal.
7596 }
7597 \@@_msg_new:nn { hvlines-except-corners }
7598 {
7599   Obsolete-key.\
7600   The-key~'hvlines-except-corners'~is-now-obsolete.~You-should-instead-use-the~
7601   keys~'hvlines'~and~'corners'.\
7602   This-error-is-fatal.
7603 }
7604 \@@_msg_new:nn { columns-not-used }
7605 {
7606   Columns-not-used.\
7607   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
7608   \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\
7609   The-columns-you-did-not-used-won't-be-created.
7610 }
7611 \@@_msg_new:nn { in-first-col }
7612 {
7613   Erroneous-use.\
7614   You-can't-use-the-command~#1 in-the-first-column-(number~0)-of-the-array.\
7615   That-command-will-be-ignored.
7616 }
7617 \@@_msg_new:nn { in-last-col }
7618 {
7619   Erroneous-use.\
7620   You-can't-use-the-command~#1 in-the-last-column-(exterior)-of-the-array.\
7621   That-command-will-be-ignored.
7622 }
7623 \@@_msg_new:nn { in-first-row }
7624 {
7625   Erroneous-use.\
7626   You-can't-use-the-command~#1 in-the-first-row-(number~0)-of-the-array.\
7627   That-command-will-be-ignored.
7628 }
7629 \@@_msg_new:nn { in-last-row }
7630 {
7631   You-can't-use-the-command~#1 in-the-last-row-(exterior)-of-the-array.\
7632   That-command-will-be-ignored.
7633 }
7634 \@@_msg_new:nn { double-closing-delimiter }
7635 {
7636   Double-delimiter.\
7637   You-can't-put-a-second-closing-delimiter~"#1"~just-after-a-first-closing~
7638   delimiter.~This-delimiter-will-be-ignored.
7639 }
7640 \@@_msg_new:nn { delimiter-after-opening }
7641 {
7642   Double-delimiter.\
7643   You-can't-put-a-second-delimiter~"#1"~just-after-a-first-opening~
7644   delimiter.~That-delimiter-will-be-ignored.

```

```

7645 }
7646 \@@_msg_new:nn { bad-option-for-line-style }
7647 {
7648   Bad-line-style.\
7649   Since-you-haven't-loaded-Tikz,~the-only-value-you-can-give-to~'line-style'~
7650   is~'standard'.~That-key-will-be-ignored.
7651 }
7652 \@@_msg_new:nnn { Unknown-key-for-custom-line }
7653 {
7654   Unknown-key.\
7655   The-key~'\l_keys_key_str'~is-unknown~in-a~'custom-line'.~
7656   It-will-be-ignored. \
7657   \c_@@_available_keys_str
7658 }
7659 {
7660   The-available-keys-are~(in~alphabetic-order):~
7661   color,~
7662   command,~
7663   dotted,~
7664   letter,~
7665   multiplicity,~
7666   sep-color,~
7667   tikz,~and~total-width.
7668 }
7669 \@@_msg_new:nn { Unknown-key-for-xdots }
7670 {
7671   Unknown-key.\
7672   As~for~now,~there~is~only~five~keys~available~here:~'color',~'inter',~
7673   'line-style',~'radius',~
7674   and~'shorten'~(and-you-try~to~use~'\l_keys_key_str').~
7675   That-key-will-be-ignored.
7676 }
7677 \@@_msg_new:nn { Unknown-key-for-rowcolors }
7678 {
7679   Unknown-key.\
7680   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
7681   (and-you-try~to~use~'\l_keys_key_str')\
7682   That-key-will-be-ignored.
7683 }
7684 \@@_msg_new:nn { Construct-too-large }
7685 {
7686   Construct-too-large.\
7687   Your-command~\token_to_str:N #1
7688   can't-be-drawn~because~your~matrix-is~too~small.\
7689   That-command-will-be-ignored.
7690 }
7691 \@@_msg_new:nn { ampersand-in-light-syntax }
7692 {
7693   Ampersand-forbidden.\
7694   You-can't-use-an-ampersand~(\token_to_str:N &)~to~separate~columns~because~
7695   the-key~'light-syntax'~is~in~force.~This-error-is-fatal.
7696 }
7697 \@@_msg_new:nn { double-backslash-in-light-syntax }
7698 {
7699   Double-backslash-forbidden.\
7700   You-can't-use~\token_to_str:N
7701   \~to~separate~rows~because~the-key~'light-syntax'~
7702   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
7703   (set~by~the-key~'end-of-row').~This-error-is-fatal.
7704 }

```

```

7705 \@@_msg_new:nn { bad-value-for-baseline }
7706 {
7707   Bad-value-for-baseline.\\
7708   The-value-given-to-'baseline'~(\int_use:N \l_tmpa_int)~is-not~
7709   valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int~ and~
7710   \int_use:N \g_@@_row_total_int~ or-equal-to-'t',~'c'~or~'b'~or-of~
7711   the-form-'line-i'.\\
7712   A-value-of-1~will-be-used.
7713 }
7714 \@@_msg_new:nn { Invalid-name }
7715 {
7716   Invalid-name.\\
7717   You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
7718   \SubMatrix~ of-your~\@@_full_name_env:.\\
7719   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
7720   This-key-will-be-ignored.
7721 }
7722 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
7723 {
7724   Wrong-line.\\
7725   You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
7726   \token_to_str:N \SubMatrix~ of-your~\@@_full_name_env:\ but-that~
7727   number-is-not-valid.~It-will-be-ignored.
7728 }
7729 \@@_msg_new:nn { Impossible-delimiter }
7730 {
7731   Impossible-delimiter.\\
7732   It's-impossible-to-draw-the~#1~delimiter~of-your~
7733   \token_to_str:N \SubMatrix~ because~all~the~cells~are~empty~
7734   in~that~column.
7735   \bool_if:NT \l_@@_submatrix_slim_bool
7736   { ~Maybe-you-should-try-without-the-key~'slim'. } \\
7737   This~\token_to_str:N \SubMatrix~ will-be-ignored.
7738 }
7739 \@@_msg_new:nn { width-without-X-columns }
7740 {
7741   No-X-column.\\
7742   You-have-used-the-key~'width'~but-you-have-put-no~'X'~column. \\
7743   That-key-will-be-ignored.
7744 }
7745 \@@_msg_new:nn { key-multiplicity-with-dotted }
7746 {
7747   Incompatible-keys. \\
7748   You-have-used-the-key~'multiplicity'~with~the-key~'dotted'~
7749   in~a~'custom-line'.~They-are-incompatible. \\
7750   The-key~'multiplicity'~will-be-discarded.
7751 }
7752 \@@_msg_new:nn { empty-environment }
7753 {
7754   Empty-environment.\\
7755   Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
7756 }
7757 \@@_msg_new:nn { Wrong-use-of-v-center }
7758 {
7759   Wrong-use-of-v-center.\\
7760   You-should-not-use-the-key~'v-center'~here~because~your~block~is~not~
7761   mono-row.~However,~you~can~go~on.
7762 }
7763 \@@_msg_new:nn { No-letter-and-no-command }
7764 {

```

```

7765     Erroneous-use.\\
7766     Your-use-of-'custom-line'~is-no-op~since~you~don't~have~used~the~
7767     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~key~'command'~
7768     (to~draw~horizontal~rules).\\
7769     However,~you~can~go~on.
7770 }

7771 \@@_msg_new:nn { Forbidden~letter }
7772 {
7773     Forbidden~letter.\\
7774     You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
7775     It~will~be~ignored.
7776 }

7777 \@@_msg_new:nn { Several~letters }
7778 {
7779     Wrong~name.\\
7780     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
7781     have~used~'\l_@@_letter_str').\\
7782     It~will~be~ignored.
7783 }

7784 \@@_msg_new:nn { Delimiter~with~small }
7785 {
7786     Delimiter~forbidden.\\
7787     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
7788     because~the~key~'small'~is~in~force.\\
7789     This~error~is~fatal.
7790 }

7791 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
7792 {
7793     Unknown~cell.\\
7794     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
7795     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\\
7796     can't~be~executed~because~a~cell~doesn't~exist.\\
7797     This~command~\token_to_str:N \line\ will~be~ignored.
7798 }

7799 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
7800 {
7801     Duplicate~name.\\
7802     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
7803     in~this~\@@_full_name_env:~\\
7804     This~key~will~be~ignored.\\
7805     \bool_if:NF \c_@@_messages_for_Overleaf_bool
7806     { For~a~list~of~the~names~already~used,~type~H~<return>. }
7807 }
7808 {
7809     The~names~already~defined~in~this~\@@_full_name_env:~ are:~
7810     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
7811 }

7812 \@@_msg_new:nn { r-or-l-with-preamble }
7813 {
7814     Erroneous-use.\\
7815     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
7816     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
7817     your~\@@_full_name_env:~\\
7818     This~key~will~be~ignored.
7819 }

7820 \@@_msg_new:nn { Hdotsfor~in~col-0 }
7821 {
7822     Erroneous-use.\\
7823     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
7824     the~array.~This~error~is~fatal.
7825 }

```



```

7826 \@@_msg_new:nn { bad-corner }
7827 {
7828   Bad-corner.\
7829   #1-is-an-incorrect-specification-for-a-corner~(in-the-key~
7830   'corners').~The-available-values-are:~NW,~SW,~NE-and-SE.\
7831   This-specification-of-corner-will-be-ignored.
7832 }
7833 \@@_msg_new:nn { bad-border }
7834 {
7835   Bad-border.\
7836   \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
7837   (in-the-key~'borders'~of-the-command~\token_to_str:N \Block).~
7838   The-available-values-are:~left,~right,~top-and-bottom~(and-you-can~
7839   also-use-the-key~'tikz'
7840   \bool_if:nF \c_@@_tikz_loaded_bool
7841   {~if-you-load-the-LaTeX-package-'tikz'}).~
7842   This-specification-of-border-will-be-ignored.
7843 }
7844 \@@_msg_new:nn { tikz-key~without~tikz }
7845 {
7846   Tikz-not-loaded.\
7847   You-can't-use-the-key~'tikz'~for-the-command~'\token_to_str:N
7848   \Block'~because-you-have-not-loaded-Tikz.~
7849   This-key-will-be-ignored.
7850 }
7851 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
7852 {
7853   Erroneous-use.\
7854   In-the~\@@_full_name_env:,~you-must-use-the-key~
7855   'last-col'~without-value.\
7856   However,~you-can-go-on-for~this-time~
7857   (the-value~'\l_keys_value_tl'~will-be-ignored).
7858 }
7859 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
7860 {
7861   Erroneous-use.\
7862   In~\NiceMatrixoptions,~you-must-use-the-key~
7863   'last-col'~without-value.\
7864   However,~you-can-go-on-for~this-time~
7865   (the-value~'\l_keys_value_tl'~will-be-ignored).
7866 }
7867 \@@_msg_new:nn { Block~too~large~1 }
7868 {
7869   Block-too-large.\
7870   You-try-to-draw-a-block-in-the-cell~#1-#2-of-your-matrix-but-the-matrix-is~
7871   too-small-for-that-block. \
7872 }
7873 \@@_msg_new:nn { Block~too~large~2 }
7874 {
7875   Block-too-large.\
7876   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7877   \g_@@_static_num_of_col_int\
7878   columns~but~you-use-only~\int_use:N \c@jCol\ and-that's-why-a-block~
7879   specified-in-the-cell~#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
7880   (&)~at~the-end-of~the-first-row-of~your~
7881   \@@_full_name_env:.\
7882   This-block-and-maybe-others-will-be-ignored.
7883 }
7884 \@@_msg_new:nn { unknown~column~type }
7885 {
7886   Bad-column-type.\

```

```

7887 The~column~type~'#1'~in~your~\@@_full_name_env:\
7888 is~unknown. \\
7889 This~error~is~fatal.
7890 }

7891 \@@_msg_new:nn { tabularnote~forbidden }
7892 {
7893   Forbidden~command.\\
7894   You~can't~use~the~command~\token_to_str:N\tabularnote\
7895   ~in~a~\@@_full_name_env:~This~command~is~available~only~in~
7896   \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
7897   This~command~will~be~ignored.
7898 }

7899 \@@_msg_new:nn { borders~forbidden }
7900 {
7901   Forbidden~key.\\
7902   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
7903   because~the~option~'rounded-corners'~
7904   is~in~force~with~a~non-zero~value.\\
7905   This~key~will~be~ignored.
7906 }

7907 \@@_msg_new:nn { bottomrule~without~booktabs }
7908 {
7909   booktabs~not~loaded.\\
7910   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
7911   loaded~'booktabs'.\\
7912   This~key~will~be~ignored.
7913 }

7914 \@@_msg_new:nn { enumitem~not~loaded }
7915 {
7916   enumitem~not~loaded.\\
7917   You~can't~use~the~command~\token_to_str:N\tabularnote\
7918   ~because~you~haven't~loaded~'enumitem'.\\
7919   This~command~will~be~ignored.
7920 }

7921 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
7922 {
7923   Tikz~not~loaded.\\
7924   You~have~used~the~key~'tikz'~in~the~definition~of~a~
7925   customized~line~(with~'custom~line')~but~Tikz~is~not~loaded.~
7926   You~can~go~on~but~you~will~have~another~error~if~you~actually~
7927   use~that~custom~line.
7928 }

7929 \@@_msg_new:nn { tikz~in~borders~without~tikz }
7930 {
7931   Tikz~not~loaded.\\
7932   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
7933   command~\token_to_str:N\Block')~but~Tikz~is~not~loaded.~
7934   That~key~will~be~ignored.
7935 }

7936 \@@_msg_new:nn { color~in~custom~line~with~tikz }
7937 {
7938   Erroneous~use.\\
7939   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
7940   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
7941   The~key~'color'~will~be~discarded.
7942 }

7943 \@@_msg_new:nn { Wrong~last~row }
7944 {
7945   Wrong~number.\\
7946   You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~

```

```

7947 \@@_full_name_env:\ seems-to-have-\int_use:N \c@iRow \ rows.~
7948 If-you-go-on,~the-value-of-\int_use:N \c@iRow \ will-be-used-for~
7949 last-row.~You-can-avoid-this-problem-by-using~'last-row'~
7950 without-value~(more-compilations-might-be-necessary).
7951 }
7952 \@@_msg_new:nn { Yet-in-env }
7953 {
7954   Nested-environments.\\
7955   Environments-of-nicematrix-can't-be-nested.\\
7956   This-error-is-fatal.
7957 }
7958 \@@_msg_new:nn { Outside-math-mode }
7959 {
7960   Outside-math-mode.\\
7961   The-\@@_full_name_env:\ can-be-used-only-in-math-mode~
7962   (and-not-in-\token_to_str:N \vcenter).\\
7963   This-error-is-fatal.
7964 }
7965 \@@_msg_new:nn { One-letter-allowed }
7966 {
7967   Bad-name.\\
7968   The-value-of-key~'\l_keys_key_str'~must-be-of-length-1.\\
7969   It-will-be-ignored.
7970 }
7971 \@@_msg_new:nn { varwidth-not-loaded }
7972 {
7973   varwidth-not-loaded.\\
7974   You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
7975   loaded.\\
7976   Your-column-will-behave-like~'p'.
7977 }
7978 \@@_msg_new:nnn { Unknown-key-for-Block }
7979 {
7980   Unknown-key.\\
7981   The-key~'\l_keys_key_str'~is-unknown-for-the-command-\token_to_str:N
7982   \Block.\\ It-will-be-ignored. \\
7983   \c_@@_available_keys_str
7984 }
7985 {
7986   The-available-keys-are~(in-alphabetic-order):~b,~borders,~c,~draw,~fill,~
7987   hlines,~hvlines,~l,~line-width,~name,~rounded-corners,~r,~respect-arraystretch,
7988   ~t,~tikz~and~vlines.
7989 }
7990 \@@_msg_new:nn { Version-of-siunitx-too-old }
7991 {
7992   siunitx-too-old.\\
7993   You-can't-use~'S'~columns-because-your-version-of~'siunitx'~
7994   is-too-old.~You-need-at-least~v3.0.\\
7995   This-error-is-fatal.
7996 }
7997 \@@_msg_new:nnn { Unknown-key-for-Brace }
7998 {
7999   Unknown-key.\\
8000   The-key~'\l_keys_key_str'~is-unknown-for-the-commands-\token_to_str:N
8001   \UnderBrace\ and-\token_to_str:N \OverBrace.\\
8002   It-will-be-ignored. \\
8003   \c_@@_available_keys_str
8004 }
8005 {
8006   The-available-keys-are~(in-alphabetic-order):~color,~left-shorten,~
8007   right-shorten,~shorten~(which-fixes-both-left-shorten-and~

```

```

8008     right-shorten)~and~yshift.
8009 }
8010 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
8011 {
8012     Unknown~key.\\
8013     The~key~'\l_keys_key_str'~is~unknown.\\
8014     It~will~be~ignored. \\
8015     \c_@@_available_keys_str
8016 }
8017 {
8018     The~available~keys~are~(in~alphabetic~order):~
8019     delimiters/color,~
8020     rules~(with~the~subkeys~'color'~and~'width'),~
8021     sub-matrix~(several~subkeys)~
8022     and~xdots~(several~subkeys).~
8023     The~latter~is~for~the~command~\token_to_str:N \line.
8024 }
8025 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
8026 {
8027     Unknown~key.\\
8028     The~key~'\l_keys_key_str'~is~unknown.\\
8029     That~key~will~be~ignored. \\
8030     \c_@@_available_keys_str
8031 }
8032 {
8033     The~available~keys~are~(in~alphabetic~order):~
8034     'delimiters/color',~
8035     'extra-height',~
8036     'hlines',~
8037     'hvlines',~
8038     'left-xshift',~
8039     'name',~
8040     'right-xshift',~
8041     'rules'~(with~the~subkeys~'color'~and~'width'),~
8042     'slim',~
8043     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
8044     and~'right-xshift').\\
8045 }
8046 \@@_msg_new:nnn { Unknown~key~for~notes }
8047 {
8048     Unknown~key.\\
8049     The~key~'\l_keys_key_str'~is~unknown.\\
8050     That~key~will~be~ignored. \\
8051     \c_@@_available_keys_str
8052 }
8053 {
8054     The~available~keys~are~(in~alphabetic~order):~
8055     bottomrule,~
8056     code-after,~
8057     code-before,~
8058     detect-duplicates,~
8059     enumitem-keys,~
8060     enumitem-keys-para,~
8061     para,~
8062     label-in-list,~
8063     label-in-tabular~and~
8064     style.
8065 }
8066 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
8067 {
8068     Unknown~key.\\
8069     The~key~'\l_keys_key_str'~is~unknown~for~the~command~

```

```

8070 \token_to_str:N \RowStyle. \\
8071 That~key~will~be~ignored. \\
8072 \c_@@_available_keys_str
8073 }
8074 {
8075   The~available~keys~are~(in~alphabetic~order):~
8076   'bold',~
8077   'cell-space-top-limit',~
8078   'cell-space-bottom-limit',~
8079   'cell-space-limits',~
8080   'color',~
8081   'nb-rows'~and~
8082   'rowcolor'.
8083 }
8084 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
8085 {
8086   Unknown~key.\\
8087   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
8088   \token_to_str:N \NiceMatrixOptions. \\
8089   That~key~will~be~ignored. \\
8090   \c_@@_available_keys_str
8091 }
8092 {
8093   The~available~keys~are~(in~alphabetic~order):~
8094   allow-duplicate-names,~
8095   cell-space-bottom-limit,~
8096   cell-space-limits,~
8097   cell-space-top-limit,~
8098   code-for-first-col,~
8099   code-for-first-row,~
8100   code-for-last-col,~
8101   code-for-last-row,~
8102   corners,~
8103   custom-key,~
8104   create-extra-nodes,~
8105   create-medium-nodes,~
8106   create-large-nodes,~
8107   delimiters~(several~subkeys),~
8108   end-of-row,~
8109   first-col,~
8110   first-row,~
8111   hlines,~
8112   hvlines,~
8113   last-col,~
8114   last-row,~
8115   left-margin,~
8116   light-syntax,~
8117   notes~(several~subkeys),~
8118   nullify-dots,~
8119   renew-dots,~
8120   renew-matrix,~
8121   respect-arraystretch,~
8122   right-margin,~
8123   rules~(with~the~subkeys~'color'~and~'width'),~
8124   small,~
8125   sub-matrix~(several~subkeys),
8126   vlines,~
8127   xdots~(several~subkeys).
8128 }
8129 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
8130 {
8131   Unknown~key.\\
8132   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~

```

```

8133 \{NiceArray\}. \\
8134 That~key~will~be~ignored. \\
8135 \c_@@_available_keys_str
8136 }
8137 {
8138 The~available~keys~are~(in~alphabetic~order):~
8139 b,~
8140 baseline,~
8141 c,~
8142 cell-space-bottom-limit,~
8143 cell-space-limits,~
8144 cell-space-top-limit,~
8145 code-after,~
8146 code-for-first-col,~
8147 code-for-first-row,~
8148 code-for-last-col,~
8149 code-for-last-row,~
8150 colortbl-like,~
8151 columns-width,~
8152 corners,~
8153 create-extra-nodes,~
8154 create-medium-nodes,~
8155 create-large-nodes,~
8156 delimiters/color,~
8157 extra-left-margin,~
8158 extra-right-margin,~
8159 first-col,~
8160 first-row,~
8161 hlines,~
8162 hvlines,~
8163 last-col,~
8164 last-row,~
8165 left-margin,~
8166 light-syntax,~
8167 name,~
8168 notes/bottomrule,~
8169 notes/para,~
8170 nullify-dots,~
8171 renew-dots,~
8172 respect-arraystretch,~
8173 right-margin,~
8174 rules~(with~the~subkeys~'color'~and~'width'),~
8175 small,~
8176 t,~
8177 tabularnote,~
8178 vlines,~
8179 xdots/color,~
8180 xdots/shorten-start,~
8181 xdots/shorten-end,~
8182 xdots/shorten-and~
8183 xdots/line-style.
8184 }

```

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).

```

8185 \@@_msg_new:nnn { Unknown-key-for-NiceMatrix }
8186 {
8187   Unknown~key.\\
8188   The~key~'\l_keys_key_str'~is~unknown~for~the~
8189   \@@_full_name_env:. \\
8190   That~key~will~be~ignored. \\
8191   \c_@@_available_keys_str
8192 }
8193 {

```

```

8194 The~available~keys~are~(in~alphabetic~order):~
8195 b,~
8196 baseline,~
8197 c,~
8198 cell-space-bottom-limit,~
8199 cell-space-limits,~
8200 cell-space-top-limit,~
8201 code-after,~
8202 code-for-first-col,~
8203 code-for-first-row,~
8204 code-for-last-col,~
8205 code-for-last-row,~
8206 colortbl-like,~
8207 columns-width,~
8208 corners,~
8209 create-extra-nodes,~
8210 create-medium-nodes,~
8211 create-large-nodes,~
8212 delimiters~(several~subkeys),~
8213 extra-left-margin,~
8214 extra-right-margin,~
8215 first-col,~
8216 first-row,~
8217 hlines,~
8218 hvlines,~
8219 l,~
8220 last-col,~
8221 last-row,~
8222 left-margin,~
8223 light-syntax,~
8224 name,~
8225 nullify-dots,~
8226 r,~
8227 renew-dots,~
8228 respect-arraystretch,~
8229 right-margin,~
8230 rules~(with~the~subkeys~'color'~and~'width'),~
8231 small,~
8232 t,~
8233 vlines,~
8234 xdots/color,~
8235 xdots/shorten-start,~
8236 xdots/shorten-end,~
8237 xdots/shorten~and~
8238 xdots/line-style.
8239 }
8240 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
8241 {
8242   Unknown~key.\\
8243   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
8244   \{NiceTabular\}. \\
8245   That~key~will~be~ignored. \\
8246   \c_@@_available_keys_str
8247 }
8248 {
8249   The~available~keys~are~(in~alphabetic~order):~
8250   b,~
8251   baseline,~
8252   c,~
8253   cell-space-bottom-limit,~
8254   cell-space-limits,~
8255   cell-space-top-limit,~
8256   code-after,~

```

```

8257 code-for-first-col,~
8258 code-for-first-row,~
8259 code-for-last-col,~
8260 code-for-last-row,~
8261 colortbl-like,~
8262 columns-width,~
8263 corners,~
8264 custom-line,~
8265 create-extra-nodes,~
8266 create-medium-nodes,~
8267 create-large-nodes,~
8268 extra-left-margin,~
8269 extra-right-margin,~
8270 first-col,~
8271 first-row,~
8272 hlines,~
8273 hvlines,~
8274 last-col,~
8275 last-row,~
8276 left-margin,~
8277 light-syntax,~
8278 name,~
8279 notes/bottomrule,~
8280 notes/para,~
8281 nullify-dots,~
8282 renew-dots,~
8283 respect-arraystretch,~
8284 right-margin,~
8285 rules~(with~the~subkeys~'color'~and~'width'),~
8286 t,~
8287 tabularnote,~
8288 vlines,~
8289 xdots/color,~
8290 xdots/shorten-start,~
8291 xdots/shorten-end,~
8292 xdots/shorten-and~
8293 xdots/line-style.
8294 }

8295 \@@_msg_new:nnn { Duplicate-name }
8296 {
8297   Duplicate-name.\
8298   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
8299   the~same~environment~name~twice.~You~can~go~on,~but,~
8300   maybe,~you~will~have~incorrect~results~especially~
8301   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
8302   message~again,~use~the~key~'allow-duplicate-names'~in~
8303   '\token_to_str:N \NiceMatrixOptions'.\
8304   \c_@@_available_keys_str
8305 }
8306 {
8307   The~names~already~defined~in~this~document~are:~
8308   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
8309 }

8310 \@@_msg_new:nn { Option-auto-for-columns-width }
8311 {
8312   Erroneous~use.\
8313   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
8314   That~key~will~be~ignored.
8315 }

```


20 History

The successive versions of the file `nicematrix.sty` provided by TeXLive are available on the svn server of TeXLive:

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

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types L, C and R.

Changes between version 1.2 and 1.3

New environment `{pNiceArrayC}` and its variants.
Correction of a bug in the definition of `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` (in fact, it was a typo).
Options are now available locally in `{pNiceMatrix}` and its variants.
The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types `w` and `W` can now be used in the environments `{NiceArray}`, `{pNiceArrayC}` and its variants with the same meaning as in the package `array`.
New option `columns-width` to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of `nicematrix` were focused on the continuous dotted lines whereas the version 2.0 of `nicematrix` provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment `{pNiceArrayRC}`. With this new implementation, there is no restriction on the width of the columns.
The package `nicematrix` no longer loads `mathtools` but only `amsmath`.
Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option `code-for-first-row` is now available in the command `\NiceMatrixOptions`.
Following a discussion on TeX StackExchange⁷², Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁷³

⁷²cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁷³Before 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`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & \ddots & \\ 0 & & 0 \end{pmatrix} L_i$$

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 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 StackExchange⁷⁴, 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).

⁷⁴cf. 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`.

Changes between versions 5.10 and 5.11

It's now possible, in the `code-before` and in the `\CodeAfter`, to use the syntax `|(i-|j)` for the Tikz node at the intersection of the (potential) horizontal rule number i and the (potential) vertical rule number j .

Changes between versions 5.11 and 5.12

Keywords `\CodeBefore` and `\Body` (alternative syntax to the key `code-before`).

New key `delimiters/max-width`.

New keys `hlines`, `vlines` and `hvlines` for the command `\SubMatrix` in the `\CodeAfter`.

New key `rounded-corners` for the command `\Block`.

Changes between versions 5.12 and 5.13

New command `\arraycolor` in the `\CodeBefore` (with its key `except-corners`).

New key `borders` for the command `\Block`.

New command `\Hline` (for horizontal rules not drawn in the blocks).

The keys `vlines` and `hlines` takes in as value a (comma-separated) list of numbers (for the rules to draw).

Changes between versions 5.13 and 5.14

Nodes of the form `(1.5)`, `(2.5)`, `(3.5)`, etc.

Keys `t` and `b` for the command `\Block`.

Key `corners`.

Changes between versions 5.14 and 5.15

Key `hvlines` for the command `\Block`.

The commands provided by `nicematrix` to color cells, rows and columns don't color the cells which are in the "corners" (when the key `corner` is used).

It's now possible to specify delimiters for submatrices in the preamble of an environment.

The version 5.15b is compatible with the version 3.0+ of `siunitx` (previous versions were not).

Changes between versions 5.15 and 5.16

It's now possible to use the cells corresponding to the contents of the nodes (of the form `i-j`) in the `\CodeBefore` when the key `create-cell-nodes` of that `\CodeBefore` is used. The medium and the large nodes are also available if the corresponding keys are used.

Changes between versions 5.16 and 5.17

The key `define-L-C-R` (only available at load-time) now raises a (non fatal) error.

Keys `L`, `C` and `R` for the command `\Block`.

Key `hvlines-except-borders`.

It's now possible to use a key `l`, `r` or `c` with the command `\pAutoNiceMatrix` (and the similar ones).

Changes between versions 5.17 and 5.18

New command `\RowStyle`

Changes between versions 5.18 and 5.19

New key `tikz` for the command `\Block`.

Changes between versions 5.19 and 6.0

Columns `X` and environment `{NiceTabularX}`.

Command `\rowlistcolors` available in the `\CodeBefore`.

In columns with fixed width, the blocks are composed as paragraphs (wrapping of the lines).

The key `define-L-C-R` has been deleted.

Changes between versions 6.0 and 6.1

Better computation of the widths of the `X` columns.

Key `\color` for the command `\RowStyle`.

Changes between versions 6.1 and 6.2

Better compatibility with the classes `revtex4-1` and `revtex4-2`.

Key `vlines-in-sub-matrix`.

Changes between versions 6.2 and 6.3

Keys `nb-rows`, `rowcolor` and `bold` for the command `\RowStyle`

Key `name` for the command `\Block`.

Support for the columns `V` of `varwidth`.

Changes between versions 6.3 and 6.4

New commands `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

Correction of a bug of the key `baseline` (cf. question 623258 on TeX StackExchange).

Correction of a bug with the columns `V` of `varwidth`.

Correction of a bug: the use of `\hdottedline` and `:` in the preamble of the array (of another letter specified by `letter-for-dotted-lines`) was incompatible with the key `xdots/line-style`.

Changes between versions 6.4 and 6.5

Key `custom-line` in `\NiceMatrixOptions`.

Key `respect-arraystretch`.

Changes between version 6.5 and 6.6

Keys `tikz` and `width` in `custom-line`.

Changes between version 6.6 and 6.7

Key `color` for `\OverBrace` and `\UnderBrace` in the `\CodeAfter`

Key `tikz` in the key `borders` of a command `\Block`

Changes between version 6.7 and 6.8

In the notes of a tabular (with the command `\tabularnote`), the duplicates are now detected: when several commands `\tabularnote` are used with the same argument, only one note is created at the end of the tabular (but all the labels are present, of course).

Changes between version 6.8 and 6.9

New keys `xdots/radius` and `xdots/inter` for customisation of the continuous dotted lines.

New command `\ShowCellNames` available in the `\CodeBefore` and in the `\CodeAfter`.

Changes between version 6.9 and 6.10

New keys `xdots/shorten-start` and `xdots/shorten-end`.

It's possible to use `\line` in the `\CodeAfter` between two blocks (and not only two cells).

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

	Symbols		
@@ commands:		<code>\l_@@_NiceTabular_bool</code> ... 186, 248, 913, 1101, 1408, 1411, 1539, 1677, 1681, 1775, 1787, 2788, 3006, 3026, 3035, 5893, 5995, 6697	
<code>\@@_Block:</code>	1262, 5811	<code>\@@_NotEmpty:</code>	1268, 2998
<code>\@@_Block_i</code>	5816, 5817, 5821	<code>\@@_OnlyMainNiceMatrix:n</code>	1264, 4784
<code>\@@_Block_ii:nnnnn</code>	5821, 5822	<code>\@@_OnlyMainNiceMatrix_i:n</code>	4787, 4794, 4797
<code>\@@_Block_iv:nnnnn</code>	5859, 5863	<code>\@@_OverBrace</code>	3152, 7248
<code>\@@_Block_iv:nnnnnn</code>	6092, 6094	<code>\@@_RowStyle:n</code>	1269, 4380
<code>\@@_Block_v:nnnnn</code>	5860, 5986	<code>\@@_S:</code>	217, 1817
<code>\@@_Block_v:nnnnnn</code>	6121, 6124	<code>\@@_ShowCellNames</code>	1401, 3153, 7400
<code>\@@_Cdots</code>	1183, 1254, 4028	<code>\@@_SubMatrix</code>	3150, 6938
<code>\g_@@_Cdots_lines_tl</code>	1285, 3218	<code>\@@_SubMatrix_in_code_before</code> ...	1400, 6918
<code>\@@_CodeAfter:</code>	1266, 6785	<code>\@@_UnderBrace</code>	3151, 7243
<code>\@@_CodeAfter_i:</code>	908, 2861, 2906, 6786	<code>\@@_V:</code>	1732, 1813
<code>\@@_CodeAfter_ii:n</code> ..	6785, 6786, 6787, 6797	<code>\@@_Vdots</code>	1184, 1255, 4044
<code>\@@_CodeAfter_iv:n</code>	6790, 6792	<code>\g_@@_Vdots_lines_tl</code>	1287, 3215
<code>\@@_CodeAfter_keys:</code>	3159, 3178	<code>\@@_Vdotsfor:</code>	1261, 4219
<code>\@@_CodeBefore:w</code>	1429, 1431	<code>\@@_Vdotsfor:nnnn</code>	4223, 4234
<code>\@@_CodeBefore_Body:w</code>	1351, 1578	<code>\@@_W:</code>	1731, 1816, 2310
<code>\@@_CodeBefore_keys:</code>	1409, 1426	<code>\@@_X</code>	1825, 3021
<code>\@@_Ddots</code>	1185, 1256, 4060	<code>\l_@@_X_column_bool</code>	253, 2210, 5857
<code>\g_@@_Ddots_lines_tl</code>	1288, 3216	<code>\l_@@_X_columns_aux_bool</code> ..	282, 1597, 2199
<code>\g_@@_HVdotsfor_lines_tl</code>	1290, 3214, 4144, 4221, 7552	<code>\l_@@_X_columns_dim</code>	283, 1598, 1607, 1613, 2202
<code>\@@_Hdotsfor:</code>	1188, 1260, 4120	<code>\@@_actually_color:</code>	1410, 4469
<code>\@@_Hdotsfor:nnnn</code>	4146, 4159	<code>\@@_actually_diagbox:nnnnnn</code>	5870, 6213, 6709, 6726
<code>\@@_Hdotsfor_i</code>	4129, 4135, 4142	<code>\@@_actually_draw_Cdots:</code>	3584, 3588
<code>\@@_Hline:</code>	1258, 5219	<code>\@@_actually_draw_Ddots:</code>	3734, 3738
<code>\@@_Hline_i:n</code>	5219, 5220, 5229	<code>\@@_actually_draw_Iddots:</code>	3782, 3786
<code>\@@_Hline_ii:nn</code>	5225, 5229	<code>\@@_actually_draw_Ldots:</code> ..	3545, 3549, 4210
<code>\@@_Hline_iii:n</code>	5226, 5230	<code>\@@_actually_draw_Vdots:</code> ..	3666, 3670, 4285
<code>\@@_Hspace:</code>	1259, 4114	<code>\@@_add_to_colors_seq:nn</code>	4455, 4467, 4468, 4492, 4501, 4510, 4519, 4623
<code>\@@_Iddots</code>	1186, 1257, 4084	<code>\@@_adjust_pos_of_blocks_seq:</code> ..	3134, 3180
<code>\g_@@_Iddots_lines_tl</code>	1289, 3217	<code>\@@_adjust_pos_of_blocks_seq_i:nnnnn</code> ..	3183, 3185
<code>\@@_Ldots</code>	1182, 1187, 1253, 4012	<code>\@@_adjust_size_box:</code>	986, 1013, 2061, 2370, 2885, 2930
<code>\g_@@_Ldots_lines_tl</code>	1286, 3219	<code>\@@_adjust_to_submatrix:nn</code>	3429, 3532, 3571, 3652, 3727, 3775
<code>\l_@@_Matrix_bool</code>	251, 1704, 1728, 2453, 2799, 2805, 2813, 2974	<code>\@@_adjust_to_submatrix:nnnnnn</code> ..	3436, 3438
<code>\l_@@_NiceArray_bool</code>	247, 416, 1332, 1572, 1644, 1767, 1774, 1786, 2453, 2787, 2799, 2805, 2813, 2950, 5003, 5007, 5165, 5175, 5205, 5209, 6689	<code>\@@_after_array:</code>	1713, 3039
<code>\g_@@_NiceMatrixBlock_int</code>	229, 5569, 5574, 5577, 5588	<code>\g_@@_after_col_zero_bool</code>	284, 1150, 2862, 4126

\@@_analyze_end:Nn	2604, 2651	\@@_cell_end:	1007, 1853,
\l_@@_argspec_tl		2000, 2057, 2088, 2218, 2321, 2350, 2366, 2983	
.....	27, 4010, 4011, 4012, 4028,	\l_@@_cell_space_bottom_limit_dim ...	
	4044, 4060, 4084, 4140, 4141, 4142, 4217,	511, 614, 1017, 4428
	4218, 4219, 4301, 4302, 4303, 6936, 6937, 6938	\l_@@_cell_space_top_limit_dim	
\@@_array:	1099, 2605, 2633	510, 612, 1015, 4417
\@@_arraycolor	1397, 4554	\@@_cellcolor ..	1391, 4536, 4548, 4549, 4755
\c_@@_arydshln_loaded_bool	36, 37	\@@_cellcolor_tabular	1192, 4749
\l_@@_auto_columns_width_bool		\g_@@_cells_seq	2644, 2645, 2646, 2648
.....	540, 704, 2718, 2722, 5564	\@@_center_cell_box:	1959, 2004
\g_@@_aux_found_bool		\@@_chessboardcolors	1399, 4541
.....	1302, 1307, 1433, 1560, 1563	\@@_cline	159, 1252
\g_@@_aux_tl	254,	\@@_cline_i:nn	160, 161, 181, 184
	1566, 1595, 1720, 3048, 3062, 3070, 3167, 5465	\@@_cline_i:w	161, 162
\c_@@_available_keys_str		\@@_cline_ii:w	166, 168
....	7524, 7526, 7657, 7983, 8003, 8015,	\@@_cline_iii:w	165, 169, 170
	8030, 8051, 8072, 8090, 8135, 8191, 8246, 8304	\l_@@_code_before_bool	
\l_@@_bar_at_end_of_pream_bool	288, 694, 721, 1123, 1322, 1354, 1569,
.....	1768, 1900, 2791		2665, 2682, 2700, 2731, 2757, 2794, 2833, 3172
\l_@@_baseline_tl	529, 530, 697, 698, 699,	\g_@@_code_before_tl ..	1558, 1567, 1570, 3169
	700, 1108, 1646, 2406, 2418, 2423, 2425,	\l_@@_code_before_tl	
	2430, 2435, 2525, 2526, 2530, 2535, 2537, 2542	287, 693, 1353, 1409, 1570
\@@_begin_of_NiceMatrix:nn	2972, 2986, 2994	\l_@@_code_for_first_col_tl	631, 2875
\@@_begin_of_row:	911, 936, 1213, 2863	\l_@@_code_for_first_row_tl ..	635, 924, 6311
\l_@@_block_auto_columns_width_bool ..		\l_@@_code_for_last_col_tl	633, 2919
.....	1552, 2723, 5557, 5562, 5572, 5582	\l_@@_code_for_last_row_tl ..	637, 931, 6314
\g_@@_block_box_int	329, 1533,	\l_@@_code_tl	275, 6913, 7149
	5865, 5879, 5881, 5939, 5951, 5963, 5972, 5982	\l_@@_col_max_int	
\l_@@_block_name_str	311, 3295, 3306, 3374, 3434, 3451
.....	6079, 6160, 6237, 6240, 6245	\l_@@_col_min_int	
\@@_block_tikz:nnnnn	6201, 6597	310, 3300, 3363, 3368, 3432, 3449
\g_@@_blocks_dp_dim		\g_@@_col_total_int	
.....	241, 994, 997, 998, 5966, 5969	234, 1030, 1281, 1362, 1373,
\g_@@_blocks_ht_dim			1446, 1630, 2291, 2292, 2750, 2751, 2781,
.....	240, 1000, 1003, 1004, 5957, 5960		2836, 2841, 2848, 2850, 2909, 3043, 3045,
\g_@@_blocks_seq			3057, 3619, 3637, 3697, 4280, 4774, 5615,
.....	299, 1554, 2463, 5976, 5988, 6092		5625, 5659, 5746, 6104, 6349, 6974, 7023, 7273
\g_@@_blocks_wd_dim		\l_@@_col_width_dim	
.....	239, 988, 991, 992, 5945, 5948	231, 232, 1980, 2050, 5898, 5903
\c_@@_booktabs_loaded_bool ..	39, 40, 1202, 2496	\l_@@_color_bool	5272, 5278, 5312, 5328
\l_@@_borders_clist	317, 6051,	\@@_color_index:n	4613, 4634, 4637
	6174, 6505, 6530, 6532, 6534, 6536, 6573, 6592	\l_@@_color_int	
\l_@@_borders_tikz_tl	4577, 4578, 4595, 4596, 4616, 4627
.....	6502, 6543, 6559, 6566, 6579, 6586	\l_@@_color_tl	
\@@_brace:nnnnn	7246, 7251, 7267		319, 4610, 4611, 4621, 4624, 5806, 5883, 5885
\l_@@_brace_left_shorten_bool		\g_@@_colors_seq ..	1405, 4458, 4462, 4463, 4473
.....	7255, 7286, 7301	\l_@@_colors_seq ..	4572, 4573, 4617, 4636, 4638
\l_@@_brace_right_shorten_bool		\@@_colortbl_like:	1190, 1272
.....	7257, 7307, 7322	\l_@@_colortbl_like_bool ..	508, 720, 1272, 1756
\l_@@_brace_yshift_dim ...	7260, 7343, 7375	\l_@@_colortbl_loaded_bool ..	113, 117, 1221
\@@_cartesian_color:nn	4482, 4494, 4503, 4625	\l_@@_cols_tl	
\@@_cartesian_path:	4486, 4724		4485, 4533, 4564, 4574, 4575, 4625, 4672, 4675
\@@_cartesian_path:n	4534, 4666, 4724	\g_@@_cols_vlism_seq	
\@@_cell_begin:w	905, 1851,	262, 1320, 1751, 1831, 7051
	1981, 2052, 2083, 2209, 2319, 2340, 2361, 2983	\@@_columncolor	1398, 4497
\l_@@_cell_box	912, 960, 962, 968, 974, 977,	\@@_columncolor_preamble	1194, 4772
	981, 990, 991, 996, 997, 1002, 1003, 1014,	\c_@@_columncolor_regex	72, 1759
	1015, 1016, 1017, 1019, 1022, 1026, 1028,	\l_@@_columns_width_dim	
	1047, 1062, 1064, 1071, 1072, 1085, 1204,	230, 705, 830, 2719, 2725, 5570, 5576
	1315, 1317, 2009, 2013, 2017, 2020, 2051,	\g_@@_com_or_env_str	266, 267, 270
	2062, 2212, 2360, 2371, 2864, 2888, 2891,	\l_@@_command_str	
	2893, 2910, 2933, 2937, 6221, 6342, 6379, 6704	5250, 5255, 5265, 5304, 5343, 5360

<code>\@@_computations_for_large_nodes: ...</code>	<code>\g_@@_dp_last_row_dim</code>
..... 5686, 5699, 5704 939, 940, 1240, 1241, 1316, 1317, 1663
<code>\@@_computations_for_medium_nodes: ..</code>	<code>\g_@@_dp_row_zero_dim</code>
..... 5606, 5675, 5685, 5696 959, 960, 1231, 1232, 1656, 2519, 2558
<code>\@@_compute_a_corner:nnnnnn</code>	<code>\@@_draw_Cdots:nnn</code>
..... 5453, 5455, 5457, 5459, 5472 3569
<code>\@@_compute_corners:</code>	<code>\@@_draw_Ddots:nnn</code>
..... 3133, 5445 3725
<code>\@@_compute_i_j:nn</code>	<code>\@@_draw_Iddots:nnn</code>
..... 6947, 6971, 7270 3773
<code>\@@_compute_rule_width:n ..</code>	<code>\@@_draw_Ldots:nnn</code>
..... 5347, 5363, 5387 3530
<code>\@@_construct_preamble:</code>	<code>\@@_draw_Vdots:nnn</code>
..... 1329, 1725 3650
<code>\l_@@_corners_cells_seq</code>	<code>\@@_draw_blocks:</code>
..... 303, 4669, 4709, 4879, 4885, 4892, 2463, 6089
5067, 5073, 5080, 5447, 5463, 5467, 5468, 5528	<code>\@@_draw_dotted_lines:</code>
<code>\l_@@_corners_clist</code> 3132, 3203
..... 534, 687, 4853, 5041, 5448	<code>\@@_draw_dotted_lines_i:</code>
<code>\@@_create_blocks_nodes:</code> 3206, 3210
..... 1382, 1478	<code>\l_@@_draw_first_bool ..</code>
<code>\@@_create_col_nodes:</code> 326, 4075, 4099, 4110
..... 2609, 2637, 2657	<code>\@@_draw_hlines:</code>
<code>\@@_create_diag_nodes: ...</code> 3135, 5201
..... 1379, 3078, 3236	<code>\@@_draw_line:</code>
<code>\@@_create_extra_nodes: ..</code> 3567,
..... 1476, 2462, 5596 3612, 3723, 3771, 3819, 3821, 4350, 4979, 5181
<code>\@@_create_large_nodes:</code>	<code>\@@_draw_line_ii:nn</code>
..... 5604, 5680 4330, 4334
<code>\@@_create_medium_and_large_nodes: ..</code>	<code>\@@_draw_line_iii:nn</code>
..... 5601, 5691 4337, 4341
<code>\@@_create_medium_nodes:</code>	<code>\@@_draw_standard_dotted_line: ..</code>
..... 5602, 5670 3828, 3860
<code>\@@_create_nodes: 5677, 5688, 5698, 5701, 5742</code>	<code>\@@_draw_standard_dotted_line_i: 3923, 3927</code>
<code>\@@_create_one_block_node:nnnnn</code>	<code>\l_@@_draw_tl</code>
..... 1484, 1487 315, 6045,
<code>\@@_create_row_node:</code> 6049, 6162, 6394, 6400, 6402, 6404, 6441, 6442
..... 1111, 1153, 1203	<code>\@@_draw_unstandard_dotted_line: 3829, 3831</code>
<code>\@@_create_row_node_i:</code>	<code>\@@_draw_unstandard_dotted_line:n ...</code>
..... 1116, 1119 3834, 3837, 3844
<code>\@@_custom_line:n</code>	<code>\@@_draw_unstandard_dotted_line:nnn ..</code>
..... 605, 5248, 5400 3839, 3845, 3859
<code>\l_@@_custom_line_commands_seq</code>	<code>\@@_draw_vlines:</code>
..... 279, 1270, 5360 3136, 4999
<code>\@@_custom_line_i:n</code>	<code>\g_@@_empty_cell_bool ..</code>
..... 5257, 5268 296, 1021, 1031,
<code>\@@_cut_on_hyphen:w</code> 2898, 2945, 4026, 4042, 4058, 4082, 4105, 4116
..... 349,	<code>\l_@@_end_int</code>
362, 4526, 4531, 4591, 4679, 4680, 4702, 4808,
4703, 4733, 4734, 6412, 6421, 6452, 6455, 4831, 4832, 4843, 4870, 5020, 5021, 5031, 5058
6475, 6478, 6506, 6509, 6922, 6927, 6953, 6960	<code>\l_@@_end_of_row_tl</code>
<code>\g_@@_ddots_int</code> 550, 551, 625, 2629, 2630, 7702
..... 3110, 3754, 3755	<code>\c_@@_endpgfortikzpicture_tl</code>
<code>\@@_def_env:nnn</code> 52, 57, 3207, 4338, 6521
..... 2956, 2967, 2968, 2969, 2970, 2971	<code>\c_@@_enumitem_loaded_bool</code>
<code>\@@_define_com:nnn</code> 42, 43, 389, 746, 755
..... 6673, 6681, 6682, 6683, 6684, 6685	<code>\@@_env:</code>
<code>\@@_delimiter:nnn</code> 224, 228, 945, 951,
..... 2116, 2137, 2145, 2158, 2164, 2170, 6800	1048, 1054, 1068, 1076, 1129, 1135, 1141,
<code>\l_@@_delimiters_color_tl</code>	1216, 1363, 1364, 1369, 1370, 1375, 1376,
..... 553,	1388, 1443, 1444, 1449, 1452, 1455, 1458,
768, 1422, 1669, 1670, 1687, 1688, 6779,	1467, 1468, 1473, 1474, 1500, 2666, 2669,
6838, 6839, 6866, 6888, 7201, 7202, 7225, 7226	2671, 2687, 2693, 2696, 2705, 2711, 2714,
<code>\l_@@_delimiters_max_width_bool</code>	2736, 2742, 2745, 2763, 2769, 2775, 2802,
..... 554, 766, 1692	2811, 2825, 2836, 2841, 2850, 3083, 3084,
<code>\g_@@_delta_x_one_dim</code>	3089, 3090, 3098, 3104, 3145, 3253, 3255,
..... 3112, 3757, 3767	3262, 3264, 3265, 3270, 3273, 3335, 3403,
<code>\g_@@_delta_x_two_dim</code>	3468, 3479, 3492, 3495, 3514, 3517, 3622,
..... 3114, 3805, 3815	3625, 3640, 3643, 4173, 4191, 4248, 4266,
<code>\g_@@_delta_y_one_dim</code>	4323, 4325, 4344, 4347, 5483, 5502, 5520,
..... 3113, 3759, 3767	5628, 5630, 5638, 5749, 5758, 5776, 6235,
<code>\g_@@_delta_y_two_dim</code>	6240, 6241, 6246, 6255, 6259, 6273, 6278,
..... 3115, 3807, 3815	6290, 6296, 6297, 6300, 6319, 6356, 6815,
<code>\@@_diagbox:nn</code>	6818, 6990, 6992, 6997, 6999, 7026, 7028,
..... 1267, 6705	7033, 7035, 7133, 7145, 7292, 7294, 7313, 7315
<code>\@@_dotfill:</code>	<code>\g_@@_env_int</code>
..... 6694 223, 224, 226, 1551, 1561, 1564, 1719, 5785
<code>\@@_dotfill_i:</code>	<code>\@@_error:n</code>
..... 6699, 6701 11,
<code>\@@_dotfill_ii:</code>	392, 417, 563, 594, 647, 762, 820, 829,
..... 6698, 6701, 6702	839, 855, 862, 870, 871, 872, 878, 883, 884,
<code>\@@_dotfill_iii:</code>	885, 899, 901, 902, 903, 1424, 1591, 1625,
..... 6702, 6703	1635, 1707, 2041, 2440, 2501, 2547, 4378,
<code>\l_@@_dotted_bool</code>	
..... 337, 3827, 4814, 4901, 4903, 5089, 5091	
<code>\l_@@_dotted_rule_bool</code>	
..... 5271, 5284, 5316, 5326, 5339, 5367, 5371	
<code>\@@_double_int_eval:n</code>	
..... 4291, 4311, 4312	
<code>\@@_double_int_eval_i:n</code>	
..... 4295, 4297	
<code>\g_@@_dp_ante_last_row_dim</code>	
..... 939, 1237	

4567, 5256, 5277, 5279, 5286, 5290, 5294,	\@@_full_name_env: 268,
5324, 6041, 6087, 6132, 6500, 6544, 6550,	7566, 7573, 7582, 7592, 7607, 7718, 7726,
6783, 6898, 6909, 6916, 7265, 7481, 7508, 7518	7755, 7787, 7795, 7803, 7809, 7815, 7817,
\@@_error:nn 12, 13, 712, 2119, 2165, 2171,	7854, 7876, 7881, 7887, 7895, 7947, 7961, 8189
2195, 4015, 4018, 4031, 4034, 4047, 4050,	\@@_h_custom_line:n 5304, 5341
4064, 4065, 4070, 4071, 4088, 4089, 4094,	\@@_h_custom_line:nn 5362
4095, 5461, 6903, 6975, 7005, 7008, 7276, 7277	\@@_hline:n 5017, 5216, 5239, 5351, 6485
\@@_error:nnn 14, 4328, 7080, 7115	\@@_hline_i: 5023, 5026
\@@_error_too_much_cols: 1796, 7537	\@@_hline_ii: 5051, 5059, 5087
\@@_everycr: 1146, 1226, 1229	\@@_hline_iii: 5095, 5099
\@@_everycr_i: 1146, 1147	\@@_hline_iv: 5092, 5152
\l_@@_except_borders_bool	\@@_hline_v: 5096, 5184
. 545, 659, 5003, 5007, 5205, 5209	\@@_hlines_block:nnn 6150, 6471
\@@_exec_code_before: 1322, 1403	\l_@@_hlines_block_bool 328, 6056, 6146, 6157
\@@_expand_clist:N 354, 1200, 1201	\l_@@_hlines_clist 333, 639, 653,
\@@_expand_clist:NN 4672, 4673, 4725	658, 1154, 1156, 1160, 1200, 3135, 5214, 5215
\l_@@_exterior_arraycolsep_bool	\l_@@_hpos_block_str 321, 322, 5790,
. 531, 826, 1777, 1789, 2790	5792, 5794, 5796, 5798, 5800, 5837, 5838,
\l_@@_extra_left_margin_dim	5840, 5892, 5904, 5917, 5928, 5979, 6003,
. 548, 675, 1345, 2896	6007, 6020, 6025, 6060, 6062, 6064, 6066,
\l_@@_extra_right_margin_dim	6069, 6072, 6323, 6335, 6346, 6350, 6360, 6372
. 549, 676, 1586, 2941, 3700	\l_@@_hpos_cell_str 237, 238,
\@@_extract_brackets 6186, 6384	1851, 1947, 1949, 2053, 2319, 2362, 5836, 5838
\@@_extract_coords_values: 5767, 5774	\l_@@_hpos_col_str
\@@_fatal:n 1903, 1905, 1907, 1909, 1934, 1946,
. 15, 258, 685, 1542, 2093, 2097, 2126,	1950, 1952, 1960, 1961, 1964, 1969, 2036, 2187
2614, 2618, 2620, 2654, 4131, 7542, 7545, 7548	\l_@@_hpos_of_block_cap_bool
\@@_fatal:nn 16, 1843, 2313 323, 6067, 6070, 6073, 6249, 6320, 6357
\l_@@_fill_tl 314, 6043, 6184, 6186	\g_@@_ht_last_row_dim
\l_@@_final_i_int 941, 1238, 1239, 1314, 1315, 1662
. 3119, 3282, 3287, 3290, 3315,	\g_@@_ht_row_one_dim 967, 968, 1235, 1236
3323, 3327, 3336, 3344, 3424, 3480, 3561,	\g_@@_ht_row_zero_dim
3634, 3640, 3643, 4164, 4192, 4260, 4270, 4272 961, 962, 1233, 1234, 1657, 2518, 2557
\l_@@_final_j_int	\@@_i: 5608, 5610,
. 3120, 3283, 3288, 3295, 3300, 3306, 3316,	5611, 5612, 5613, 5622, 5628, 5630, 5631,
3324, 3328, 3337, 3345, 3423, 3425, 3481,	5632, 5633, 5638, 5639, 5640, 5641, 5649,
3514, 3517, 3525, 4185, 4195, 4197, 4239, 4268	5652, 5654, 5655, 5656, 5708, 5710, 5713,
\l_@@_final_open_bool	5714, 5718, 5719, 5744, 5749, 5751, 5753,
. 3122, 3289, 3293, 3296, 3303,	5757, 5758, 5769, 5776, 5778, 5780, 5784, 5785
3309, 3313, 3329, 3558, 3593, 3598, 3609,	\g_@@_iddots_int 3111, 3802, 3803
3673, 3683, 3688, 3709, 3746, 3794, 3931,	\l_@@_in_env_bool 244, 416, 1542, 1543
3946, 4162, 4186, 4198, 4237, 4261, 4273, 4320	\l_@@_initial_i_int 3117,
\@@_find_extremities_of_line:nnnn	3280, 3355, 3358, 3383, 3391, 3395, 3404,
. 3277, 3535, 3574, 3655, 3730, 3778	3412, 3422, 3469, 3554, 3600, 3602, 3616,
\l_@@_first_col_int	3622, 3625, 4163, 4164, 4174, 4242, 4252, 4254
. 147, 160, 340, 341, 627, 876,	\l_@@_initial_j_int
911, 1373, 1446, 1639, 1769, 2660, 2680, 3118, 3281, 3356, 3363,
3055, 3619, 3637, 4124, 4693, 4786, 5615,	3368, 3374, 3384, 3392, 3396, 3405, 3413,
5625, 5659, 5707, 5746, 6654, 6660, 6666, 7023	3423, 3425, 3470, 3492, 3495, 3503, 3690,
\l_@@_first_i_tl 6949,	3692, 3697, 4167, 4177, 4179, 4238, 4239, 4250
6955, 6956, 6986, 7017, 7026, 7028, 7083,	\l_@@_initial_open_bool 3121, 3357, 3361,
7090, 7092, 7174, 7185, 7189, 7289, 7310, 7338	3364, 3371, 3377, 3381, 3397, 3551, 3590,
\l_@@_first_j_tl	3597, 3607, 3673, 3680, 3686, 3740, 3788,
. 6950, 6958, 6959, 6990, 6992, 7053, 7066,	3929, 4161, 4168, 4180, 4236, 4243, 4255, 4319
7073, 7075, 7175, 7186, 7190, 7292, 7294, 7304	\@@_insert_tabularnotes: 2468, 2471
\l_@@_first_row_int 338, 339, 628, 880,	\@@_instruction_of_type:nnn
1279, 1367, 1441, 1654, 2437, 2516, 2544, 1088, 4020, 4036, 4052, 4075, 4099
2555, 3052, 3489, 3511, 5608, 5622, 5649,	\c_@@_integers_alist_tl 7159, 7170
5706, 5744, 6252, 6270, 6652, 6812, 6987, 7709	\g_@@_internal_code_after_tl 276, 1887,
\c_@@_footnote_bool	2115, 2136, 2144, 2157, 2163, 2169, 3155,
. 1528, 1723, 7457, 7479, 7502, 7505, 7515, 7521	3156, 5237, 5349, 5390, 5868, 6211, 6707, 6930
\c_@@_footnotehyper_bool 7456, 7480, 7512	\@@_intersect_our_row:nnnnn 4655
\c_@@_forbidden_letters_str 5293, 5306	\@@_intersect_our_row_p:nnnnn 4605

<code>\@@_j:</code>	5615, 5617, 5618, 5619, 5620, 5625, 5628, 5630, 5633, 5635, 5636, 5638, 5641, 5643, 5644, 5659, 5662, 5664, 5665, 5666, 5721, 5723, 5726, 5728, 5732, 5733, 5746, 5749, 5750, 5752, 5757, 5758, 5770, 5776, 5777, 5779, 5784, 5785
<code>\l_@@_key_nb_rows_int</code>	236, 4372, 4373, 4383, 4394, 4402, 4409
<code>\l_@@_l_dim</code>	3907, 3908, 3921, 3922, 3934, 3940, 3951, 3960, 3970, 3975, 3982, 3985, 3992, 3995
<code>\l_@@_large_nodes_bool</code>	544, 666, 5600, 5604
<code>\g_@@_last_col_found_bool</code>	348, 1284, 1631, 1699, 2749, 2828, 2907, 3042, 6347
<code>\l_@@_last_col_int</code> 346, 347, 821, 848, 850, 863, 879, 900, 1305, 1308, 1634, 1781, 2979, 2981, 3043, 3045, 3660, 3695, 4017, 4033, 4071, 4095, 6097, 6102, 6103, 6104, 6107, 6143, 6153, 6169, 6181, 6193, 6205, 6217, 6232, 6273, 6278, 6286, 6302, 6656, 6662, 6668, 7541, 7574
<code>\l_@@_last_col_without_value_bool</code> ..	345, 847, 3044, 7544
<code>\l_@@_last_empty_column_int</code>	5493, 5494, 5507, 5513, 5526
<code>\l_@@_last_empty_row_int</code>	5475, 5476, 5489, 5510
<code>\l_@@_last_i_tl</code>	6951, 6962, 6963, 6973, 6986, 7020, 7033, 7035, 7083, 7090, 7272, 7289, 7310, 7370
<code>\l_@@_last_j_tl</code>	6952, 6965, 6966, 6974, 6997, 6999, 7056, 7066, 7073, 7273, 7313, 7315, 7325
<code>\l_@@_last_row_int</code>	342, 343, 629, 929, 975, 1166, 1299, 1303, 1310, 1619, 1623, 1626, 1638, 1660, 2631, 2632, 2871, 2872, 2916, 2917, 3047, 3540, 3579, 4049, 4065, 4089, 4792, 4800, 6096, 6099, 6100, 6119, 6143, 6153, 6169, 6181, 6193, 6204, 6216, 6230, 6301, 6313, 6664, 7946
<code>\g_@@_last_row_node_int</code>	235, 1113, 1115, 1326
<code>\l_@@_last_row_without_value_bool</code> ..	344, 1301, 1621, 3046
<code>\l_@@_left_delim_dim</code>	1330, 1334, 1339, 2593, 2894
<code>\g_@@_left_delim_tl</code>	1338, 1530, 1671, 1695, 1765, 2100, 2102, 5167
<code>\l_@@_left_margin_dim</code>	546, 669, 1344, 2895, 5164, 5737
<code>\l_@@_letter_str</code>	5251, 5254, 5262, 5287, 5289, 5293, 5298, 7774, 7781
<code>\l_@@_letter_vlism_tl</code>	261, 646, 1829
<code>\l_@@_light_syntax_bool</code>	528, 623, 1347, 1581
<code>\@@_light_syntax_i</code>	2622, 2625
<code>\@@_line</code>	3154, 4303
<code>\@@_line_i:nn</code>	4310, 4317
<code>\l_@@_line_width_dim</code>	320, 6058, 6395, 6433, 6444, 6450, 6473, 6497, 6529, 6555, 6557, 6574, 6575, 6577, 6595
<code>\@@_line_with_light_syntax:n</code> ..	2636, 2640
<code>\@@_line_with_light_syntax_i:n</code>	2635, 2641, 2642, 2650
<code>\l_@@_local_end_int</code>	4841, 4862, 4870, 4927, 4976, 4991, 5029, 5050, 5058, 5115, 5170, 5172, 5193
<code>\l_@@_local_start_int</code>	4840, 4856, 4857, 4860, 4864, 4868, 4916, 4974, 4987, 5028, 5044, 5045, 5048, 5052, 5056, 5104, 5160, 5162, 5189
<code>\@@_math_toggle_token:</code>	185, 1009, 2865, 2882, 2911, 2927, 6752, 6763, 7356, 7392
<code>\g_@@_max_cell_width_dim</code>	1018, 1019, 1553, 2724, 5563, 5589
<code>\c_@@_max_l_dim</code>	3921, 3926
<code>\l_@@_medium_nodes_bool</code>	543, 665, 5598, 6293
<code>\@@_message_hdotsfor:</code>	7550, 7573, 7582, 7592
<code>\c_@@_messages_for_Overleaf_bool</code>	20, 7455, 7478, 7523, 7805
<code>\@@_msg_new:nn</code>	17, 7484, 7493, 7555, 7562, 7568, 7577, 7587, 7597, 7604, 7611, 7617, 7623, 7629, 7634, 7640, 7646, 7669, 7677, 7684, 7691, 7697, 7705, 7714, 7722, 7729, 7739, 7745, 7752, 7757, 7763, 7771, 7777, 7784, 7791, 7812, 7820, 7826, 7833, 7844, 7851, 7859, 7867, 7873, 7884, 7891, 7899, 7907, 7914, 7921, 7929, 7936, 7943, 7952, 7958, 7965, 7971, 7990, 8310
<code>\@@_msg_new:nnn</code>	18, 7652, 7799, 7978, 7997, 8010, 8025, 8046, 8066, 8084, 8129, 8185, 8240, 8295
<code>\@@_msg_redirect_name:nn</code>	25, 832, 1711, 6110, 6113
<code>\@@_multicolumn:nnn</code>	1274, 2251
<code>\g_@@_multicolumn_cells_seq</code>	306, 1277, 1324, 2266, 3068, 3072, 3073, 5633, 5641, 5763, 6257, 6275
<code>\g_@@_multicolumn_sizes_seq</code>	307, 1278, 1325, 2268, 3074, 3075, 5764
<code>\l_@@_multiplicity_int</code>	4812, 4924, 4925, 4931, 4943, 4953, 5112, 5113, 5119, 5129, 5139, 5283, 5309, 5331, 5378, 5379
<code>\g_@@_name_env_str</code>	265, 271, 272, 1537, 1538, 2653, 2951, 2952, 2960, 2961, 2991, 3004, 3022, 3032, 3174, 6677, 7539
<code>\l_@@_name_str</code>	542, 714, 947, 950, 1050, 1053, 1137, 1140, 2670, 2671, 2695, 2696, 2713, 2714, 2744, 2745, 2771, 2774, 2821, 2824, 2843, 2847, 3092, 3097, 3103, 3254, 3255, 3266, 3269, 3272, 5754, 5757, 5781, 5784, 6242, 6245
<code>\g_@@_names_seq</code>	243, 711, 713, 8308
<code>\l_@@_nb_cols_int</code>	6630, 6641, 6649, 6655, 6661, 6667
<code>\l_@@_nb_rows_int</code>	6629, 6640, 6658
<code>\@@_newcolumnntype</code>	1173, 1730, 1731, 1732, 3010
<code>\@@_node_for_cell:</code>	1027, 1034, 1414, 2892, 2942
<code>\@@_node_for_multicolumn:nn</code>	5765, 5772
<code>\@@_node_left:nn</code>	7132, 7133, 7193
<code>\@@_node_position:</code>	1369, 1371, 1375, 1377, 1443, 1445, 1453, 1459
<code>\@@_node_position_i:</code>	1456, 1460
<code>\@@_node_right:nnnn</code>	7142, 7144, 7217
<code>\g_@@_not_empty_cell_bool</code>	286, 1025, 1032, 2999

<code>\@@_not_in_exterior:nnnnn</code>	4647	<code>\@@_patch_preamble_ii:nn</code>	1806, 1807, 1808, 1858
<code>\@@_not_in_exterior_p:nnnnn</code>	4583	<code>\@@_patch_preamble_iii:n</code>	1809, 1863, 1871
<code>\l_@@_notes_above_space_dim</code>	535, 537	<code>\@@_patch_preamble_iii_i:n</code>	1866, 1868
<code>\l_@@_notes_bottomrule_bool</code>	732, 866, 894, 2494	<code>\@@_patch_preamble_iv:n</code>	1810, 1811, 1812, 1919
<code>\l_@@_notes_code_after_tl</code>	730, 2503	<code>\@@_patch_preamble_iv_i:n</code>	1922, 1924
<code>\l_@@_notes_code_before_tl</code>	728, 2475	<code>\@@_patch_preamble_iv_ii:w</code>	1927, 1928, 1930
<code>\l_@@_notes_detect_duplicates_bool</code>	245, 246, 420, 760	<code>\@@_patch_preamble_iv_iii:nn</code>	1931, 1932
<code>\@@_notes_format:n</code>	378, 432, 437	<code>\@@_patch_preamble_iv_iv:nn</code>	1936, 1938, 2039, 2042, 2201
<code>\@@_notes_label_in_list:n</code>	385, 404, 412, 740	<code>\@@_patch_preamble_iv_v:nnnnnnnn</code>	1942, 1975
<code>\@@_notes_label_in_tabular:n</code>	384, 443, 737	<code>\@@_patch_preamble_ix:nn</code>	1821, 1822, 1823, 2124
<code>\l_@@_notes_labels_seq</code>	376, 431, 436, 446, 451	<code>\@@_patch_preamble_ix_i:nnn</code>	2128, 2150
<code>\l_@@_notes_para_bool</code>	726, 864, 892, 2479	<code>\@@_patch_preamble_v:n</code>	1813, 1814, 2025
<code>\@@_notes_style:n</code>	381, 383, 386, 404, 412, 734	<code>\@@_patch_preamble_v_i:w</code>	2028, 2029, 2031
<code>\l_@@_nullify_dots_bool</code>	538, 664, 4024, 4040, 4056, 4080, 4103	<code>\@@_patch_preamble_v_ii:nn</code>	2032, 2033
<code>\@@_old_CT@arc@</code>	1544, 3176	<code>\@@_patch_preamble_vi:nnnn</code>	1815, 1816, 2045
<code>\@@_old_cdots</code>	1246, 4041	<code>\@@_patch_preamble_vii:n</code>	1817, 2068
<code>\@@_old_ddots</code>	1248, 4081	<code>\@@_patch_preamble_vii_i:w</code>	2071, 2072, 2074
<code>\@@_old_dotfill</code>	6693, 6696, 6704	<code>\@@_patch_preamble_vii_ii:n</code>	2075, 2076
<code>\@@_old_dotfill:</code>	1265	<code>\@@_patch_preamble_viii:nn</code>	1818, 1819, 1820, 2095
<code>\l_@@_old_iRow_int</code>	277, 1295, 3223	<code>\@@_patch_preamble_viii_i:nn</code>	2108, 2111, 2113
<code>\@@_old_ialign:</code>	1110, 1242, 3149, 6091	<code>\@@_patch_preamble_x:n</code>	1824, 1825, 2175
<code>\@@_old_iddots</code>	1249, 4104	<code>\@@_patch_preamble_x_i:w</code>	2178, 2179, 2181
<code>\l_@@_old_jCol_int</code>	278, 1297, 3224	<code>\@@_patch_preamble_x_ii:n</code>	2182, 2185
<code>\@@_old_ldots</code>	1245, 4025	<code>\@@_patch_preamble_xi:n</code>	1856, 1973, 2066, 2091, 2222, 2225, 2249
<code>\@@_old_multicolumn</code>	1276, 4119	<code>\@@_patch_preamble_xiii:n</code>	2228, 2246
<code>\@@_old_pgfpntanchor</code>	199, 7151, 7155	<code>\@@_pgf_rect_node:nnn</code>	484, 1457, 6295
<code>\@@_old_pgful@check@rerun</code>	106, 110	<code>\@@_pgf_rect_node:nnnnn</code>	459, 1499, 5748, 5775, 6234, 6289, 7119
<code>\@@_old_vdots</code>	1247, 4057	<code>\c_@@_pgfortikzpicture_tl</code>	51, 56, 3205, 4336, 6519
<code>\@@_open_x_final_dim:</code>	3508, 3560, 3594, 3748, 3797	<code>\@@_pgfpntanchor:n</code>	7147, 7152
<code>\@@_open_x_initial_dim:</code>	3486, 3553, 3591, 3743, 3791	<code>\@@_pgfpntanchor_i:nn</code>	7155, 7157
<code>\@@_open_y_final_dim:</code>	3632, 3684, 3796	<code>\@@_pgfpntanchor_ii:w</code>	7158, 7166
<code>\@@_open_y_initial_dim:</code>	3614, 3681, 3742, 3790	<code>\@@_pgfpntanchor_iii:w</code>	7179, 7181
<code>\l_@@_other_keys_tl</code>	4833, 4902, 5022, 5090, 5252, 5257	<code>\@@_picture_position:</code>	1364, 1371, 1377, 1445, 1459, 1460
<code>\@@_overbrace_i:n</code>	7331, 7336	<code>\g_@@_pos_of_blocks_seq</code>	300, 1323, 1483, 1555, 2269, 3060, 3064, 3065, 3182, 4581, 4847, 5035, 5540, 6159, 6717
<code>\l_@@_parallelize_diags_bool</code>	532, 533, 661, 3108, 3752, 3800	<code>\g_@@_pos_of_stroken_blocks_seq</code>	302, 1556, 4851, 5039, 6171
<code>\@@_patch_for_revtext:</code>	1507, 1526	<code>\g_@@_pos_of_xdots_seq</code>	301, 1557, 3420, 4849, 5037
<code>\@@_patch_m_preamble:n</code>	2260, 2295, 2328, 2333, 2394	<code>\l_@@_position_int</code>	4804, 4834, 4842, 4918, 4971, 4989, 5030, 5106, 5157, 5191
<code>\@@_patch_m_preamble_i:n</code>	2299, 2300, 2301, 2315	<code>\g_@@_post_action_cell_tl</code>	907, 1011, 2006, 2211, 4415, 4426
<code>\@@_patch_m_preamble_ii:nn</code>	2302, 2303, 2304, 2325	<code>\@@_pre_array:</code>	1293, 1355, 1578
<code>\@@_patch_m_preamble_iii:n</code>	2305, 2330	<code>\@@_pre_array_ii:</code>	1196, 1327
<code>\@@_patch_m_preamble_iv:nnn</code>	2306, 2307, 2308, 2335	<code>\@@_pre_code_before:</code>	1357, 1435
<code>\@@_patch_m_preamble_ix:n</code>	2379, 2397	<code>\c_@@_preamble_first_col_tl</code>	1770, 2857
<code>\@@_patch_m_preamble_v:nnnn</code>	2309, 2310, 2355	<code>\c_@@_preamble_last_col_tl</code>	1782, 2902
<code>\@@_patch_m_preamble_x:n</code>	2323, 2353, 2374, 2376, 2400	<code>\g_@@_preamble_tl</code>	1532, 1733, 1737, 1741, 1747, 1761, 1770, 1779, 1782, 1791, 1795, 1833, 1849, 1860, 1873, 1977, 2047, 2080, 2107, 2135, 2143, 2156, 2162, 2206, 2232,
<code>\@@_patch_node_for_cell:</code>	1060, 1414		
<code>\@@_patch_node_for_cell:n</code>	1058, 1084, 1087		
<code>\@@_patch_preamble:n</code>	1753, 1799, 1835, 1841, 1861, 1897, 2104, 2120, 2122, 2138, 2146, 2172, 2243		
<code>\@@_patch_preamble_i:n</code>	1803, 1804, 1805, 1847		

2239, 2248, 2259, 2261, 2317, 2327, 2332,
 2337, 2357, 2383, 2390, 2399, 2605, 2633, 5388
 \@_provide_pgfsyspdfmark: ... 73, 82, 1527
 \@_put_box_in_flow: ... 1697, 2402, 2595
 \@_put_box_in_flow_bis:nn ... 1694, 2562
 \@_put_box_in_flow_i: ... 2408, 2410
 \@_qpoint:n ...
 227, 1491, 1493, 1495, 1497, 2413, 2415,
 2427, 2443, 2510, 2512, 2528, 2539, 2550,
 3242, 3244, 3246, 3248, 3258, 3260, 3503,
 3525, 3554, 3561, 3600, 3602, 3616, 3634,
 3690, 3692, 4344, 4347, 4692, 4696, 4712,
 4714, 4916, 4918, 4927, 4971, 4974, 4976,
 4987, 4989, 4991, 5104, 5106, 5115, 5157,
 5160, 5170, 5189, 5191, 5193, 5654, 5664,
 6226, 6228, 6230, 6232, 6266, 6286, 6316,
 6417, 6419, 6426, 6428, 6554, 6556, 6558,
 6572, 6576, 6578, 6731, 6733, 6736, 6738,
 6805, 6807, 7017, 7020, 7058, 7075, 7092,
 7304, 7325, 7338, 7370, 7408, 7410, 7419, 7421
 \l_@@_real_left_delim_dim 2564, 2579, 2594
 \l_@@_real_right_delim_dim 2565, 2591, 2597
 \@_recreate_cell_nodes: ... 1380, 1439
 \g_@@_recreate_cell_nodes_bool ...
 541, 1211, 1380, 1406, 1413, 1418
 \@_rectanglecolor ...
 1392, 4389, 4506, 4539, 4556, 4766
 \@_rectanglecolor:nnn ... 4512, 4521, 4524
 \@_renew_NC@rewriteS: ... 210, 212, 1283
 \@_renew_dots: ... 1180, 1273
 \l_@@_renew_dots_bool ... 662, 1273, 7474
 \@_renew_matrix: ... 824, 6609, 7476
 \l_@@_respect_arraystretch_bool ...
 539, 680, 5808, 5888, 5899, 5998, 6015, 6082
 \l_@@_respect_blocks_bool 4562, 4579, 4602
 \@_restore_iRow_jCol: ... 3175, 3221
 \c_@@_revtex_bool ... 61, 64, 67, 68, 1526
 \l_@@_right_delim_dim ...
 1331, 1335, 1341, 2596, 2939
 \g_@@_right_delim_tl .. 1340, 1531, 1689,
 1695, 1766, 2103, 2132, 2133, 2154, 2159, 5177
 \l_@@_right_margin_dim ...
 547, 671, 1585, 2940, 3699, 5174, 5740
 \@_rotate: ... 1263, 4290
 \g_@@_rotate_bool ...
 252, 984, 1012, 1997, 2060, 2369, 2884,
 2929, 4290, 5892, 5936, 5941, 6002, 6019, 6222
 \@_rotate_cell_box: ...
 972, 1012, 2060, 2369, 2884, 2929, 6222
 \l_@@_rounded_corners_dim ...
 318, 6047, 6194, 6409, 6410, 6445, 6499, 6593
 \@_roundedrectanglecolor ... 1393, 4515
 \l_@@_row_max_int ... 309, 3290, 3433, 3450
 \l_@@_row_min_int ... 308, 3358, 3431, 3448
 \g_@@_row_of_col_done_bool ...
 285, 1151, 1536, 2679
 \g_@@_row_style_tl ...
 289, 919, 1559, 1989, 4407, 4408,
 4410, 4413, 4424, 4435, 4440, 4451, 4452, 5886
 \g_@@_row_total_int ... 233, 1280, 1361,
 1367, 1441, 1637, 2438, 2545, 3047, 3054,
 3489, 3511, 4205, 5608, 5622, 5649, 5744,
 6119, 6252, 6270, 6812, 6973, 6987, 7272, 7710
 \@_rowcolor ... 1394, 4398, 4488
 \@_rowcolor_tabular ... 1193, 4760
 \@_rowcolors ... 1395, 4640
 \@_rowcolors_i:nnnn ... 4606, 4642
 \l_@@_rowcolors_restart_bool ... 4565, 4594
 \@_rowlistcolors ... 1396, 4569, 4641
 \g_@@_rows_seq . 2628, 2630, 2632, 2634, 2636
 \l_@@_rows_tl ...
 4484, 4532, 4608, 4625, 4673, 4698
 \l_@@_rule_width_dim ...
 250, 4824, 4922, 4972, 4990, 5110, 5158,
 5192, 5335, 5348, 5355, 5372, 5376, 5389, 5396
 \l_@@_rules_color_tl ...
 280, 598, 1576, 1577, 7048, 7049
 \@_set_CT@arc@: ... 187, 1577, 4817, 7049
 \@_set_CT@arc@_i: ... 188, 189
 \@_set_CT@arc@_ii: ... 188, 191
 \@_set_CT@drsc@: ... 193, 4819
 \@_set_CT@drsc@_i: ... 194, 195
 \@_set_CT@drsc@_ii: ... 194, 197
 \@_set_final_coords: ... 3459, 3484
 \@_set_final_coords_from_anchor:n ..
 3475, 3564, 3595, 3676, 3685, 3751, 3799
 \@_set_initial_coords: ... 3454, 3473
 \@_set_initial_coords_from_anchor:n ..
 3464, 3557, 3592, 3675, 3682, 3745, 3793
 \@_set_size:n ... 6627, 6642
 \l_@@_siunitx_loaded_bool ... 200, 204, 209
 \g_@@_size_seq ...
 1303, 1308, 1359, 1360, 1361, 1362, 3050
 \l_@@_small_bool ... 822, 853, 859,
 881, 916, 1206, 2097, 2126, 2866, 2912, 3123
 \@_standard_cline ... 144, 1251
 \@_standard_cline:w ... 144, 145
 \l_@@_standard_cline_bool .. 509, 610, 1250
 \c_@@_standard_tl ... 526, 527, 3826
 \l_@@_start_int ... 4806, 4843, 5031
 \g_@@_static_num_of_col_int ...
 313, 1706, 1754, 6107, 7593, 7608, 7877
 \l_@@_stop_loop_bool ... 3284, 3285,
 3317, 3330, 3339, 3352, 3353, 3385, 3398, 3407
 \@_store_in_tmpb_tl ... 6387, 6389
 \@_stroke_block:nnn ... 6166, 6391
 \@_stroke_borders_block:nnn ... 6178, 6495
 \@_stroke_borders_block_i: ... 6512, 6517
 \@_stroke_borders_block_ii: ... 6520, 6524
 \@_stroke_horizontal:n .. 6535, 6537, 6570
 \@_stroke_vertical:n ... 6531, 6533, 6552
 \@_sub_matrix:nnnnnnn ... 6942, 6968
 \@_sub_matrix_i:nnnn ... 7009, 7015
 \l_@@_submatrix_extra_height_dim ...
 330, 6858, 7043
 \l_@@_submatrix_hlines_clist ...
 335, 6870, 6890, 7082, 7084
 \l_@@_submatrix_left_xshift_dim ...
 331, 6860, 7095, 7128
 \l_@@_submatrix_name_str ...
 6905, 6977, 7117, 7119, 7131, 7133, 7141, 7145
 \g_@@_submatrix_names_seq ...
 304, 3158, 6902, 6906, 7810
 \l_@@_submatrix_right_xshift_dim ...
 332, 6862, 7104, 7138
 \g_@@_submatrix_seq ... 312, 1321, 3435, 6928

<code>\l_@@_submatrix_slim_bool</code>	6868, 6985, 7735	<code>\\@@_use_arraybox_with_notes:</code>	... 1651, 2523
<code>\l_@@_submatrix_vlines_clist</code>	<code>\\@@_use_arraybox_with_notes_b:</code>	.. 1648, 2507
.....	336, 6872, 6892, 7065, 7067	<code>\\@@_use_arraybox_with_notes_c:</code>
<code>\l_@@_suffix_tl</code> 5676, 5687,	1649, 1680, 2451, 2521, 2560
.....	5697, 5700, 5749, 5757, 5758, 5776, 5784, 5785	<code>\l_@@_v_center_bool</code>	.. 6084, 6128, 6133, 6308
<code>\l_@@_tabular_width_dim</code>	<code>\\@@_v_custom_line:n</code> 5299, 5385
.....	249, 1104, 1106, 1793, 3033	<code>\\c_@@_varwidth_loaded_bool</code>	... 33, 34, 2038
<code>\l_@@_tabularnote_tl</code>	375, 868, 896, 2467, 2476	<code>\\@@_vline:n</code> 1889, 4828, 5014, 5392, 6462
<code>\\g_@@_tabularnotes_seq</code>	<code>\\@@_vline_i:</code> 4835, 4838
.....	374, 422, 433, 2466, 2482, 2488, 2504	<code>\\@@_vline_ii:</code> 4863, 4871, 4899
<code>\\c_@@_tabularx_loaded_bool</code>	... 45, 46, 3021	<code>\\@@_vline_iii:</code> 4907, 4911
<code>\\@@_test_hline_in_block:nnnnn</code>	<code>\\@@_vline_iv:</code> 4904, 4966
.....	5036, 5038, 5401	<code>\\@@_vline_v:</code> 4908, 4982
<code>\\@@_test_hline_in_stroken_block:nnnn</code>	..	<code>\\@@_vlines_block:nnn</code> 6140, 6448
.....	5040, 5423	<code>\\l_@@_vlines_block_bool</code>	327, 6054, 6136, 6157
<code>\\@@_test_if_cell_in_a_block:nn</code>	<code>\\l_@@_vlines_clist</code> 334, 640,
.....	5479, 5497, 5515, 5535	652, 657, 1201, 1739, 1745, 1776, 1788,
<code>\\@@_test_if_cell_in_block:nnnnnnn</code>	2230, 2237, 2381, 2388, 2789, 3136, 5012, 5013
.....	5541, 5543	<code>\\l_@@_vpos_col_str</code>
<code>\\@@_test_if_math_mode:</code> 255, 1541, 2962	1911, 1914, 1916, 1921, 1943, 1959, 2035, 2188
<code>\\@@_test_in_corner_h:</code> 5041, 5062	<code>\\l_@@_vpos_of_block_tl</code>	.. 324, 325, 5802,
<code>\\@@_test_in_corner_v:</code> 4853, 4874	5804, 5902, 5916, 5927, 6006, 6024, 6075, 6077
<code>\\@@_test_vline_in_block:nnnnn</code>	<code>\\@@_w:</code> 1730, 1815, 2309
.....	4848, 4850, 5412	<code>\\l_@@_weight_int</code>
<code>\\@@_test_vline_in_stroken_block:nnnn</code>	2184, 2189, 2190, 2193, 2195, 2196, 2198, 2202
.....	4852, 5434	<code>\\l_@@_width_dim</code> 242,
<code>\\l_@@_the_array_box</code>	817, 889, 1603, 1611, 3002, 3003, 3023, 3024
.....	1328, 1343, 1603, 1611, 2455, 2456, 2458, 2461	<code>\\g_@@_width_first_col_dim</code>
<code>\\c_@@_tikz_loaded_bool</code>	298, 1535, 1642, 2674, 2887, 2888
.....	50, 55, 1383, 3137, 6039, 7840	<code>\\g_@@_width_last_col_dim</code>
<code>\\l_@@_tikz_rule_bool</code>	297, 1534, 1701, 2832, 2932, 2933
...	5270, 5274, 5314, 5327, 5334, 5365, 5374	<code>\\l_@@_width_used_bool</code> 305, 890, 1588
<code>\\l_@@_tikz_rule_tl</code>	<code>\\l_@@_x_final_dim</code>
...	4821, 4906, 4993, 4994, 5094, 5195, 5196	292, 3461, 3510, 3519, 3520, 3523,
<code>\\l_@@_tikz_seq</code> 316, 6040, 6197, 6206	3526, 3527, 3678, 3694, 3702, 3706, 3710,
<code>\\g_@@_tmpc_dim</code> 7402, 7420, 7422, 7442	3712, 3717, 3719, 3749, 3758, 3766, 3806,
<code>\\l_@@_tmpc_dim</code> 294, 1496, 1503,	3814, 3856, 3871, 3880, 3914, 3969, 3979,
.....	3247, 3251, 4694, 4695, 4718, 4928, 4939,	4348, 4973, 5171, 5174, 5176, 5178, 6984,
.....	4947, 4952, 4958, 4992, 4996, 5116, 5133,	7000, 7001, 7007, 7104, 7121, 7138, 7309,
.....	5138, 5144, 5194, 5198, 6231, 6236, 6291,	7316, 7317, 7323, 7326, 7342, 7359, 7374, 7389
.....	6420, 6431, 6557, 6562, 6567, 6737, 6740, 6748	<code>\\l_@@_x_initial_dim</code>	290, 3456, 3488, 3497,
<code>\\l_@@_tmpc_int</code> 4597, 4598, 4599	3498, 3501, 3504, 3505, 3678, 3693, 3694,
<code>\\l_@@_tmpc_tl</code>	... 4375, 4385, 4390, 4399,	3701, 3706, 3710, 3712, 3714, 3717, 3719,
.....	4527, 4529, 4532, 4691, 4710, 6453, 6465,	3758, 3766, 3806, 3814, 3853, 3870, 3880,
.....	6476, 6481, 6507, 6537, 6554, 6923, 6925, 6929	3914, 3969, 3977, 3979, 4001, 4003, 4345,
<code>\\g_@@_tmpd_dim</code> 7403, 7422, 7430	4972, 4973, 5161, 5164, 5166, 5168, 6983,
<code>\\l_@@_tmpd_dim</code> 295,	6993, 6994, 7004, 7095, 7120, 7128, 7288,
.....	1498, 1504, 3249, 3252, 4715, 4718, 4940,	7295, 7296, 7302, 7305, 7342, 7359, 7374, 7389
.....	4947, 5126, 5133, 6233, 6236, 6269, 6280,	<code>\\l_@@_xdots_color_tl</code>	552, 566, 3544, 3583,
.....	6284, 6287, 6291, 6429, 6432, 6739, 6740, 6758	3664, 3665, 3733, 3781, 3835, 4209, 4284, 4307
<code>\\l_@@_tmpd_tl</code>	4528, 4530, 4533, 6454, 6458,	<code>\\l_@@_xdots_down_tl</code>	... 591, 3842, 3864, 3899
.....	6477, 6488, 6508, 6533, 6572, 6924, 6926, 6929	<code>\\l_@@_xdots_inter_dim</code>
<code>\\g_@@_tmpe_dim</code> 7404, 7423, 7442	512, 514, 589, 3126, 3934, 3941,
<code>\\g_@@_total_X_weight_int</code>	3952, 3963, 3970, 3975, 3982, 3992, 5168, 5178
.....	281, 1199, 1590, 1593, 1612, 2198	<code>\\l_@@_xdots_line_style_tl</code>
<code>\\l_@@_total_width_bool</code>	... 5336, 5366, 5369	525, 527, 562, 3826, 3835
<code>\\l_@@_type_of_col_tl</code> 851,	<code>\\l_@@_xdots_radius_dim</code>
.....	852, 2992, 2994, 6634, 6635, 6636, 6644, 6649	522, 524, 585, 3125, 3565, 3566, 4002, 5372
<code>\\g_@@_types_of_matrix_seq</code>	<code>\\l_@@_xdots_shorten_end_dim</code>
.....	7529, 7530, 7535, 7539	516, 520, 572,
<code>\\@@_underbrace_i:n</code> 7330, 7368	579, 3129, 3130, 3850, 3951, 3961, 3983, 3993
<code>\\@@_update_for_first_and_last_row:</code>	..		
.....	955, 1020, 1312, 2886, 2931		

$\backslash l_@@_xdots_shorten_start_dim$	2688, 2694, 2704, 2706, 2712, 2735, 2737, 2743, 2761, 2764, 2770, 2798, 2804, 2816, 2819, 4694, 4695, 4697, 4713, 4715, 4924, 4938, 4939, 4942, 4955, 4961, 5014, 5112, 5128, 5141, 5147, 5216, 5234, 5378, 5589, 6395, 6450, 6473, 6490, 6497, 6830, 7043, 7047
..... 515, 519, 571, 576, 3127, 3128, 3851, 3940, 3961, 3983, 3993	
$\backslash l_@@_xdots_up_tl$ 592, 3841, 3863, 3889	
$\backslash l_@@_y_final_dim$	
..... 293, 3462, 3562, 3566, 3604, 3608, 3610, 3635, 3645, 3646, 3760, 3763, 3808, 3811, 3856, 3871, 3879, 3916, 3974, 3989, 4349, 4977, 5159, 6808, 6830, 6845, 7021, 7036, 7037, 7042, 7060, 7077, 7121, 7129, 7139	
$\backslash l_@@_y_initial_dim$ 291, 3457, 3555, 3565, 3603, 3604, 3608, 3610, 3617, 3627, 3628, 3760, 3765, 3808, 3813, 3853, 3870, 3879, 3916, 3974, 3987, 3989, 4001, 4004, 4346, 4975, 5158, 5159, 6806, 6830, 6845, 7018, 7029, 7030, 7042, 7059, 7076, 7120, 7129, 7139	
$\backslash \backslash$ 21, 2619, 2641, 6656, 6662, 6668, 6786, 7460, 7461, 7490, 7499, 7557, 7559, 7564, 7570, 7579, 7589, 7599, 7601, 7606, 7608, 7613, 7614, 7619, 7620, 7625, 7626, 7631, 7636, 7642, 7648, 7654, 7656, 7671, 7679, 7681, 7686, 7688, 7693, 7699, 7701, 7707, 7711, 7716, 7718, 7719, 7724, 7731, 7736, 7741, 7742, 7747, 7749, 7754, 7759, 7765, 7768, 7773, 7774, 7779, 7781, 7786, 7788, 7793, 7796, 7801, 7803, 7804, 7814, 7817, 7822, 7828, 7830, 7835, 7841, 7846, 7853, 7855, 7861, 7863, 7869, 7871, 7875, 7881, 7886, 7888, 7893, 7896, 7901, 7904, 7909, 7911, 7916, 7918, 7923, 7931, 7938, 7945, 7954, 7955, 7960, 7962, 7967, 7968, 7973, 7975, 7980, 7982, 7992, 7994, 7999, 8001, 8002, 8012, 8013, 8014, 8027, 8028, 8029, 8044, 8048, 8049, 8050, 8068, 8070, 8071, 8086, 8088, 8089, 8131, 8133, 8134, 8187, 8189, 8190, 8242, 8244, 8245, 8297, 8303, 8312	
$\backslash \{$ 272, 1820, 2117, 2142, 2969, 6685, 7100, 7794, 7896, 8133, 8244	
$\backslash \}$ 272, 1823, 2117, 2127, 2969, 6685, 7109, 7794, 7896, 8133, 8244	
$\backslash \sqcup$ 7553, 7573, 7582, 7592, 7593, 7607, 7608, 7709, 7710, 7718, 7726, 7733, 7737, 7755, 7787, 7795, 7797, 7802, 7809, 7823, 7876, 7877, 7878, 7887, 7894, 7902, 7917, 7947, 7948, 7961, 8001	
$\backslash $ 2971, 6684	
A	
$\backslash A$	6900
$\backslash aboveulesep$	2498
$\backslash addtocounter$	449
$\backslash alpha$	383
$\backslash anchor$	3233, 3234
$\backslash array$	1515
$\backslash arraybackslash$	1990, 2213, 2343
$\backslash arraycolor$	1397
$\backslash arraycolsep$	
.. 670, 672, 674, 1103, 1209, 1334, 1335, 1679, 1683, 2456, 2800, 2806, 2816, 5166, 5176	
$\backslash arrayrulecolor$	120
$\backslash arrayrulewidth$	
152, 157, 177, 600, 946, 1127, 1130, 1136, 1167, 1674, 1685, 1742, 1748, 1834, 1881, 2233, 2240, 2384, 2391, 2515, 2554, 2686,	
B	
$\backslash baselineskip$	123, 129, 2018
$\backslash begingroup$	2254
$\backslash bfseries$	4444, 4447
$\backslash bgroup$	1529
$\backslash Block$	1262, 7837, 7848, 7902, 7933, 7982
$\backslash BNiceMatrix$	6624
$\backslash bNiceMatrix$	6621
$\backslash Body$	1351
$\backslash boldmath$	4444, 4447
bool commands:	
$\backslash bool_const:Nn$	33, 34, 36, 37, 39, 40, 42, 43, 45, 46, 50, 55, 61, 64, 67, 68
$\backslash bool_do_until:Nn$	3285, 3353
$\backslash bool_gset_false:N$	984, 1031, 1032, 1150, 1284, 1406, 1536, 1560, 1738, 2898, 2945, 5410, 5421, 5432, 5443, 5941
$\backslash bool_gset_true:N$	
1563, 1896, 2679, 2862, 2907, 2999, 4026, 4042, 4058, 4082, 4105, 4116, 4290, 4846, 5034	
$\backslash bool_if:NtF$	186, 420, 746, 755, 913, 916, 1012, 1123, 1151, 1202, 1206, 1211, 1272, 1273, 1302, 1307, 1322, 1380, 1383, 1408, 1411, 1413, 1433, 1526, 1528, 1542, 1552, 1588, 1621, 1699, 1704, 1723, 1728, 1756, 1768, 1997, 2060, 2097, 2126, 2369, 2494, 2665, 2682, 2700, 2718, 2731, 2757, 2794, 2828, 2833, 2866, 2884, 2912, 2929, 3021, 3042, 3044, 3046, 3108, 3123, 3137, 3607, 3609, 3752, 3800, 4024, 4040, 4056, 4080, 4103, 4438, 4579, 4602, 5165, 5175, 5274, 5278, 5369, 5374, 5488, 5506, 5524, 5572, 5582, 5604, 5888, 5892, 5936, 5998, 6002, 6015, 6019, 6128, 6136, 6146, 6222, 6249, 6293, 6320, 6357, 6697, 7286, 7307, 7502, 7512, 7544, 7735, 7805
$\backslash bool_if:nTF$	209, 389, 416, 1090, 1631, 3440, 4321, 4657, 5003, 5007, 5205, 5209, 5281, 5598, 5849, 6156, 6307, 6347, 6809, 6819, 6821, 6834, 6840, 6849, 7840
$\backslash bool_lazy_all:nTF$	1772, 1784, 2785, 4929, 5117, 5403, 5414, 5425, 5436, 5895
$\backslash bool_lazy_and:nnTF$	
.... 2453, 2721, 2799, 2805, 2813, 2867, 3596, 3862, 4122, 4668, 5253, 6413, 7069, 7086	
$\backslash bool_lazy_or:nnTF$ 559, 1024, 1082, 1764, 2436, 2465, 2543, 2915, 3673, 3825, 3920, 4649, 4681, 4685, 4735, 4739, 5480, 5498, 5516, 5824, 5829, 6972, 7271, 7300, 7321	
$\backslash bool_lazy_or:p:nn$	2870
$\backslash bool_not_p:n$	1775, 1777, 1787, 1789, 2723, 2788, 2790, 7301, 7322
$\backslash bool_set:Nn$	3677

\c_false_bool	34, 37, 40, 43, 46, 55, 68, 2137, 2145, 2158, 2164, 2170, 4020, 4036, 4052
\g_tmpa_bool	4846, 4854, 4881, 4889, 4894, 5034, 5042, 5069, 5077, 5082, 5410, 5421, 5432, 5443
\g_tmpb_bool	1738, 1768, 1896
\l_tmpb_bool	5485, 5499, 5517, 5539, 5552
\c_true_bool	33, 36, 39, 42, 45, 50, 61, 64, 67, 2116
box commands:	
\box_clear_new:N	1204, 1328
\box_dp:N	940, 960, 997, 1017, 1072, 1232, 1241, 1317, 2348, 2405, 3636, 5971, 7022
\box_gclear_new:N	5878
\box_grotate:Nn	5938
\box_ht:N	941, 962, 968, 980, 1003, 1015, 1064, 1234, 1236, 1239, 1315, 1985, 2009, 2011, 2017, 2344, 2404, 3618, 5962, 7019
\box_ht_plus_dp:N	2573, 2586, 7431
\box_move_down:nn	1072, 2015
\box_move_up:nn	89, 91, 93, 1064, 2448, 2521, 2560
\box_rotate:Nn	974
\box_scale:Nnn	7435
\box_set_dp:Nn	996, 1016, 2405
\box_set_ht:Nn	1002, 1014, 2404
\box_set_wd:Nn	990, 2455
\box_use:N	452, 981, 1071, 2020, 7448
\box_use_drop:N	1022, 1028, 1047, 2062, 2371, 2407, 2448, 2449, 2461, 2893, 5981, 6342, 6379
\box_wd:N	453, 991, 1019, 1026, 1085, 1339, 1341, 1603, 1611, 2456, 2458, 2580, 2592, 2888, 2891, 2933, 2937, 5950, 6704, 7430
\l_tmpa_box	441, 452, 453, 1338, 1339, 1340, 1341, 1666, 2404, 2405, 2407, 2448, 2449, 2573, 2586, 7416, 7430, 7431, 7435, 7448
\l_tmpb_box	2566, 2580, 2581, 2592
C	
\c	72, 1760
\Cdots	1254, 4031, 4034
\cdots	1183, 1246
\cellcolor	1192, 1391
\centering	1954, 5906
char commands:	
\char_set_catcode_space:n	1716
\chessboardcolors	1399
\cline	180, 1251, 1252
clist commands:	
\clist_clear:N	358, 4728
\clist_if_empty:NTF	4853, 5041, 6174
\clist_if_empty_p:N	2789
\clist_if_in:NnTF	356, 1159, 1745, 2237, 2388, 5013, 5215, 6530, 6532, 6534, 6536, 6573
\clist_map_inline:Nn	359, 4675, 4698, 4729, 5448, 7067, 7084
\clist_map_inline:nn	2987, 4538, 4587, 6600
\clist_new:N	317, 333, 334, 335, 336, 534
\clist_put_right:Nn	368, 4746
\clist_set:Nn	652, 653, 657, 658
\clist_set_eq:NN	4727
\l_tmpa_clist	358, 368, 370, 4727, 4729
\CodeAfter	908, 1266, 2622, 2625, 2861, 2906, 3157, 7795
\CodeBefore	1524
\color	124, 130, 190, 192, 196, 198, 918, 1670, 1688, 3538, 3541, 3544, 3577, 3580, 3583, 3658, 3661, 3665, 3733, 3781, 4203, 4206, 4209, 4278, 4281, 4284, 4307, 4436, 4475, 5885, 6049, 6839, 7202, 7226, 7282, 7417
\colorlet	263, 264, 925, 932, 1198, 2876, 2920
\columncolor	1194, 1398, 3166, 4778
\cr	156, 182, 2855, 7356, 7360, 7390, 7392
\crrc	2659, 7355, 7387
cs commands:	
\cs_generate_variant:Nn	13, 28, 29, 71, 184, 2650, 2986, 3844, 3859, 4467, 4468, 5362, 5594, 5595
\cs_gset:Npn	124, 130, 5587
\cs_gset_eq:NN	82, 1225, 1544, 3176
\cs_if_exist:NTF	66, 1295, 1297, 1448, 1545, 1548, 1732, 2078, 3223, 3224, 3320, 3333, 3388, 3401, 3491, 3513, 3621, 3639, 4171, 4189, 4246, 4264, 5276, 5574, 5627, 6254, 6272, 6542, 6814, 6989, 6996, 7025, 7032, 7291, 7312
\cs_if_exist_p:N	560, 4932, 5120, 5482, 5501, 5519
\cs_if_free:NTF	78, 3012, 3533, 3572, 3653, 3728, 3776
\cs_if_free_p:N	4323, 4325
\cs_new_protected:Npx	3203, 4334, 6517
\cs_new_protected_nopar:Npn	1351
\cs_set:Nn	734, 737, 740
\cs_set:Npn	120, 121, 126, 127, 132, 144, 145, 159, 161, 162, 168, 170, 190, 192, 196, 198, 386, 1175, 1520, 1521, 2255, 2286, 3014, 3279, 3341, 3409, 4213, 4288, 5219, 5220, 5229, 5230, 5343, 5889, 5999, 6016
\cs_set_eq:NN	1271
\cs_set_nopar:Npn	1105, 1208, 1219, 5769, 5770
\cs_set_nopar:Npx	1106
\cs_set_protected:Npn	6675
\cs_set_protected_nopar:Npn	5866, 6209
D	
\Ddots	1256, 4064, 4065, 4070, 4071
\ddots	1185, 1248
\diagbox	1267, 5866, 6209
dim commands:	
\dim_add:Nn	5738
\dim_compare:nNnTF	123, 129, 988, 994, 1000, 1793, 2719, 2937, 3002, 3501, 3523, 3712, 4411, 4422, 5651, 5661, 6264, 6284, 6704
\dim_compare_p:n	5898
\dim_gzero:N	992, 998, 1004
\dim_max:nn	5640, 5644
\dim_min:nn	5632, 5636
\dim_ratio:nn	3767, 3815, 3934, 3939, 3950, 3958, 3970, 3975, 3980, 3990, 7430, 7431
\dim_set:Nn	5613, 5620, 5631, 5635, 5639, 5643, 5655, 5656, 5665, 5666, 5710, 5723
\dim_set_eq:NN	5611, 5618, 5718, 5732
\dim_sub:Nn	5735

<code>\dim_use:N</code>		F	
.... 5652, 5662, 5713, 5714, 5726, 5727, 5750, 5751, 5752, 5753, 5777, 5778, 5779, 5780		<code>\fi</code>	134, 1735, 2255, 2258, 4451, 5219, 5245
<code>\dim_zero_new:N</code>	5610, 5612, 5617, 5619	fi commands:	
<code>\c_max_dim</code>	3488, 3501, 3510, 3523, 5611, 5613, 5618, 5620, 5652, 5662, 6251, 6264, 6269, 6284, 6810, 6811, 6983, 6984, 7004, 7007, 7288, 7302, 7309, 7323	<code>\fi:</code>	259, 4448
<code>\g_tmpb_dim</code>	7412, 7431	<code>\five</code>	3228, 3233
<code>\dotfill</code>	1265, 6693	flag commands:	
<code>\dots</code>	1187	<code>\flag_clear_new:n</code>	7148
<code>\doublerulesep</code>	1882, 4925, 4942, 4956, 5113, 5128, 5142, 5235, 5379	<code>\flag_height:n</code>	7173
<code>\doublerulesepcolor</code>	126	<code>\flag_raise:n</code>	7172
<code>\downbracefill</code>	7360	<code>\fontdimen</code>	2444
<code>\draw</code>	3847, 4994, 5196, 6566, 6586	fp commands:	
		<code>\fp_eval:n</code>	3875
E		<code>\fp_min:nn</code>	7427, 7429
<code>\egroup</code>	1521, 1714	<code>\fp_set:Nn</code>	7425
<code>\else</code>	2255	<code>\fp_to_dim:n</code>	3910
else commands:		<code>\fp_use:N</code>	7435
<code>\else:</code>	257, 4446	<code>\l_tmpa_fp</code>	7425, 7435
<code>\endarray</code>	1519, 1521, 2610, 2638	<code>\futurelet</code>	139
<code>\endBNiceMatrix</code>	6625	G	
<code>\endbNiceMatrix</code>	6622	<code>\globaldefs</code>	1710, 6112
<code>\endgroup</code>	2263	group commands:	
<code>\endNiceArray</code>	3009, 3029, 3038	<code>\group_insert_after:N</code>	6698, 6699, 6701, 6702
<code>\endNiceArrayWithDelims</code>	2955, 2965	H	
<code>\endpgfscope</code>	3904, 6757	<code>\halign</code>	1243, 7353, 7385
<code>\endpNiceMatrix</code>	6613	<code>\hbox</code>	1121, 1675, 2459, 2521, 2560, 2684, 2702, 2729, 2733, 2759, 2796
<code>\endsavenotes</code>	1723	hbox commands:	
<code>\entabular</code>	1521	<code>\hbox:n</code>	89, 91, 94
<code>\entabularnotes</code>	2489	<code>\hbox_gset:Nn</code>	5880
<code>\endVNiceMatrix</code>	6619	<code>\hbox_overlap_left:n</code>	1065, 1073, 2663, 2889
<code>\endvNiceMatrix</code>	6616	<code>\hbox_overlap_right:n</code>	452, 2830, 2935
<code>\enskip</code>	2107, 2135, 2143, 2156, 2162	<code>\hbox_set:Nn</code>	441, 1062, 1338, 1340, 1666, 2013, 2212, 2566, 2581, 6221, 7416
<code>\ensuremath</code>	4025, 4041, 4057, 4081, 4104	<code>\hbox_set:Nw</code>	912, 1343, 2051, 2360, 2864, 2910
<code>\everycr</code>	155, 182, 1229, 7352, 7382	<code>\hbox_set_end:</code>	1010, 1587, 2059, 2368, 2883, 2928
<code>\everypar</code>	1983, 1986	<code>\hbox_to_wd:nn</code>	477, 502, 7358, 7388
exp commands:		<code>\Hdotsfor</code>	1260, 7553, 7823
<code>\exp_after:wN</code>		<code>\hdotsfor</code>	1188
.... 216, 1577, 1671, 1689, 1753, 2260, 7049		<code>\heavyrulewidth</code>	2499
<code>\exp_args:Nnc</code>	5576	<code>\hfil</code>	1816, 2310, 7355, 7387
<code>\exp_args:NNV</code>	4012, 4028, 4044, 4060, 4084, 4142, 4219, 4303, 6938	<code>\hfill</code>	152, 177
<code>\exp_args:NnV</code>	5292	<code>\Hline</code>	1258, 5224
<code>\exp_args:NNx</code>	1158, 2236, 2387	<code>\hline</code>	132
<code>\exp_args:NV</code>		hook commands:	
.... 1733, 2261, 2605, 2633, 4993, 5195, 5257, 6404		<code>\hook_gput_code:nnn</code>	30, 98, 114, 201, 207, 387, 513, 517, 523, 536, 569, 575, 578, 584, 588, 744, 753, 1275, 3201, 4008, 4138, 4215, 4299, 4332, 6515, 6934
<code>\exp_args:Nxx</code>	5859, 5860	<code>\hrule</code>	136, 152, 177, 1167, 2499, 6844, 7207, 7231
<code>\exp_last_unbraced:NV</code>	1409, 3159, 6186	<code>\hsize</code>	1997
<code>\exp_not:N</code>	51, 52, 56, 57, 1834, 1875, 1947, 1949, 1954, 1955, 1956, 3050, 3064, 3072, 3074, 3169, 4415, 4426, 4436, 4778, 4994, 5196, 5389, 5467, 5916, 5927, 6006, 6024, 6189, 6566, 6586, 6648, 7155	<code>\hskip</code>	135
<code>\exp_not:n</code>	1096, 1720, 3170, 4152, 4153, 4229, 4755, 4766, 4768, 4779, 5299, 5875, 5979, 5991, 5992, 6141, 6151, 6167, 6179, 6191, 6218, 6650, 6714, 6715	<code>\Hspace</code>	1259
<code>\expandafter</code>	3013	<code>\hspace</code>	4117
<code>\ExplSyntaxOff</code>	80, 1722, 5591	<code>\hss</code>	1816, 2310
<code>\ExplSyntaxOn</code>	77, 1715, 5584	I	
<code>\extrarowheight</code> ...	3618, 5890, 6000, 6017, 7019	<code>\ialign</code>	1110, 1219, 1242, 3149, 6091
		<code>\iddots</code>	1257, 4088, 4089, 4094, 4095
		<code>\iddots</code>	84, 1186, 1249
		if commands:	
		<code>\if_mode_math:</code>	257, 4442
		<code>\IfBooleanTF</code>	1578

\null 2289
 \nulldelimiterspace 2580, 2592, 6825, 7045
 \nullfont 6836, 6843, 7199, 7206, 7223, 7230

O

\omit 147, 2662, 2678, 2754, 2780, 6785, 6786
 \OnlyMainNiceMatrix 1264, 4783
 \OverBrace 3152, 7277, 8001

P

\par 2476, 2484
 \path 6602

peek commands:

\peek_meaning:NTF
 188, 194, 439, 1176, 2603, 3015, 5224
 \peek_meaning_remove_ignore_spaces:NTF 180
 \peek_remove_spaces:n 2601,
 4751, 4762, 5222, 5813, 6920, 6940, 7245, 7250
 \pgfdeclareshape 3226
 \pgfextracty 6316
 \pgfgetlastxy 495
 \pgfpathcircle 4000
 \pgfpathlineto 4952, 4958,
 5138, 5144, 6562, 6582, 6740, 7060, 7077, 7111
 \pgfpathmoveto 4951, 4957,
 5137, 5143, 6561, 6581, 6735, 7059, 7076, 7102
 \pgfpathrectanglecorners 4717, 4945, 5131, 6430
 \pgfpointadd 493
 \pgfpointanchor 199, 228, 3466,
 3477, 3494, 3516, 3624, 3642, 5630, 5638,
 6259, 6277, 6297, 6299, 6317, 6354, 6817,
 6992, 6999, 7028, 7035, 7147, 7151, 7294, 7315
 \pgfpointdiff .. 494, 1371, 1377, 1445, 1459, 1460
 \pgfpointlineattime 3869
 \pgfpointorigin 2669, 2842, 3234
 \pgfpointscale 493
 \pgfpointshapeborder 4344, 4347
 \pgfrememberpicturepositiononpagetrue ...
 943, 1038, 1134,
 1482, 2668, 2692, 2710, 2741, 2768, 2810,
 2839, 3212, 3239, 3823, 4343, 4914, 4969,
 4985, 5102, 5155, 5187, 5673, 5683, 5694,
 6224, 6398, 6526, 6730, 6803, 6980, 7284, 7437
 \pgfscope 3866, 6747
 \pgfset 462,
 487, 1039, 6331, 6368, 6746, 6824, 6982, 7328
 \pgfsetbaseline 1037
 \pgfsetcornersarced 4716, 6406
 \pgfsetlinewidth .. 4961, 5147, 6433, 6529, 7047
 \pgfsetrectcap 4962, 5148
 \pgfsetroundcap 6743
 \pgfsetstrokecolor 6404
 \pgfsyspdfmark 78, 79
 \pgftransformrotate 3873
 \pgftransformshift
 468, 493, 3250, 3867, 6330, 6352,
 6748, 6758, 6826, 7125, 7135, 7339, 7371, 7439
 \pgfusepath 3893, 3903, 4478, 4948, 6434
 \pgfusepathqfill 4006, 5134
 \pgfusepathqstroke
 4963, 5149, 6563, 6583, 6744, 7061, 7078, 7112
 \phantom 4025, 4041, 4057, 4081, 4104
 \pNiceMatrix 6612

prg commands:

\prg_do_nothing: ... 82, 210, 593, 1225, 3157
 \prg_new_conditional:Nnn 4647, 4655
 \prg_replicate:nn 2750,
 2751, 4156, 4953, 5139, 6655, 6658, 6661, 6667
 \prg_return_false: 4652, 4664
 \prg_return_true: 4653, 4663
 \ProcessKeysOptions 7483
 \ProvideDocumentCommand 84
 \ProvidesExplPackage 4

Q

\quad 410

quark commands:

\q_stop 144,
 145, 161, 162, 165, 166, 168, 169, 170,
 189, 191, 195, 197, 349, 362, 1409, 1431,
 1577, 1753, 1826, 1896, 2130, 2152, 2260,
 2311, 2622, 2625, 4291, 4295, 4297, 4311,
 4312, 4526, 4531, 4591, 4679, 4680, 4702,
 4703, 4733, 4734, 4817, 4819, 5767, 5774,
 5816, 5817, 5821, 6186, 6389, 6412, 6421,
 6452, 6455, 6475, 6478, 6506, 6509, 6627,
 6642, 6922, 6927, 6953, 6960, 7049, 7158, 7166

R

\raggedleft 1956, 5907, 6762
 \raggedright 1955, 1997, 5908
 \rectanglecolor 1392, 3165
 \refstepcounter 450

regex commands:

\regex_const:Nn 72
 \regex_match:nnTF 6900
 \regex_replace_all:NnN 1758
 \renewcommand 214
 \RenewDocumentEnvironment
 6611, 6614, 6617, 6620, 6623
 \RequirePackage 1, 3, 9, 10
 \right 1689, 2576, 2588, 6849, 7211, 7235
 \rotate 1263
 \roundedrectanglecolor 1393, 6189
 \rowcolor 1193, 1394
 \rowcolors 1395
 \rowlistcolors 1396
 \RowStyle 1269, 8070

S

\savedanchor 3228
 \savenotes 1528

scan commands:

\scan_stop: 3160
 \scriptstyle
 916, 2866, 2912, 3854, 3855, 3889, 3899

seq commands:

\seq_clear:N 451, 1751
 \seq_clear_new:N 4572, 5447
 \seq_count:N 2632, 4463, 4617
 \seq_gclear:N 1277, 1278, 1320,
 1321, 1323, 1554, 1555, 1556, 1557, 2504, 3158
 \seq_gclear_new:N 1324, 1325, 1405, 2628, 2644
 \seq_gpop_left:NN 2634, 2646
 \seq_gput_left:Nn 713, 2266, 2268, 6159
 \seq_gput_right:Nn 433, 1831, 2269,
 3420, 4462, 5976, 5988, 6171, 6717, 6906, 6928

<code>\seq_gset_from_clist:Nn</code>	3050, 3064, 3072, 3074, 7530
<code>\seq_gset_map_x:NNn</code>	3182, 7535
<code>\seq_gset_split:Nnn</code>	28, 2630, 2645
<code>\seq_if_empty:NnTF</code>	2463, 3060, 3068, 5463, 6197
<code>\seq_if_empty_p:N</code>	2466, 4669
<code>\seq_if_in:NnTF</code>	711, 4709, 4878, 4884, 4891, 5066, 5072, 5079, 5633, 5641, 6257, 6275, 6902, 7539
<code>\seq_item:Nn</code>	1303, 1308, 1359, 1360, 1361, 1362, 4636, 4638
<code>\seq_map_break:</code>	425
<code>\seq_map_function:NN</code>	2636
<code>\seq_map_indexed_inline:Nn</code>	422, 4458, 4473
<code>\seq_map_inline:Nn</code>	1270, 1483, 2482, 2488, 2648, 3435, 4606, 4847, 4849, 4851, 5035, 5037, 5039, 5540, 6092, 7051
<code>\seq_mapthread_function:NNN</code>	5762
<code>\seq_new:N</code>	243, 262, 279, 299, 300, 301, 302, 303, 304, 306, 307, 312, 316, 374, 376, 7529
<code>\seq_put_left:Nn</code>	5360
<code>\seq_put_right:Nn</code>	431, 436, 5527, 6040
<code>\seq_set_eq:NN</code>	4581
<code>\seq_set_filter:NNn</code>	4582, 4604
<code>\seq_set_from_clist:Nn</code>	5467
<code>\seq_set_split:Nnn</code>	4573
<code>\seq_use:Nn</code>	6206
<code>\seq_use:Nnnn</code>	445, 3065, 3073, 3075, 5468, 7810, 8308
<code>\l_tmpa_seq</code>	4582, 4604
<code>\l_tmpb_seq</code>	4581, 4582, 4604, 4606
<code>\setcounter</code>	380
<code>\setlist</code>	396, 407, 747, 756
<code>\ShowCellNames</code>	1401, 3153
siunitx commands:	
<code>\siunitx_cell_begin:w</code>	1960, 2078, 2085
<code>\siunitx_cell_end:</code>	1961, 2088
skip commands:	
<code>\skip_gadd:Nn</code>	2726
<code>\skip_gset:Nn</code>	2717, 2782
<code>\skip_gset_eq:NN</code>	2724, 2725
<code>\skip_horizontal:N</code>	153, 178, 1344, 1345, 1585, 1586, 1641, 1642, 1678, 1679, 1682, 1683, 1701, 1702, 1742, 1748, 1834, 2233, 2240, 2384, 2391, 2593, 2594, 2596, 2597, 2673, 2674, 2686, 2688, 2704, 2706, 2728, 2735, 2737, 2756, 2761, 2764, 2783, 2793, 2798, 2800, 2804, 2806, 2832, 2894, 2895, 2896, 2899, 2934, 2939, 2940, 2941
<code>\skip_horizontal:n</code>	453, 1085, 1877, 5389
<code>\skip_vertical:N</code>	157, 1127, 1130, 2473, 2498
<code>\skip_vertical:n</code>	980, 1674, 1685, 5232, 5348, 7357, 7391
<code>\g_tmpa_skip</code>	2717, 2724, 2725, 2726, 2728, 2756, 2782, 2783
<code>\c_zero_skip</code>	1230
<code>\smash</code>	7236, 7237
<code>\space</code>	271, 272, 7836
<code>\stepcounter</code>	430
str commands:	
<code>\c_backslash_str</code>	271
<code>\str_case:nn</code>	1952, 5904, 6323, 6335, 6360, 6372, 7096, 7105
<code>\str_case:nnTF</code>	1646, 1801, 1964, 2297, 2430, 5450, 7170, 7183
<code>\str_clear_new:N</code>	5250, 5251, 6977
<code>\str_const:Nn</code>	5306, 7524, 7526
<code>\str_count:N</code>	5289
<code>\str_gclear:N</code>	3174
<code>\str_gset:Nn</code>	1538, 2952, 2961, 2991, 3004, 3022, 3032, 6677
<code>\str_if_empty:NnTF</code>	947, 1050, 1137, 1537, 2670, 2695, 2713, 2744, 2771, 2821, 2843, 2951, 2960, 3092, 3254, 3266, 5287, 5304, 5754, 5781, 5836, 6237, 6242, 7117, 7131, 7141
<code>\str_if_empty_p:N</code>	5254, 5255
<code>\str_if_eq:nnTF</code>	109, 270, 1108, 1829, 1870, 1896, 1926, 1943, 1946, 1959, 1960, 1961, 2027, 2070, 2100, 2132, 2154, 2177, 2227, 2378, 2617, 2619, 2653, 4371, 4636, 6402, 6794, 7275, 7329
<code>\str_if_eq_p:nn</code>	561, 1765, 1766, 4683, 4687, 4737, 4741, 5826, 5831
<code>\str_if_in:NnTF</code>	2418, 2530
<code>\str_new:N</code>	237, 265, 321, 542
<code>\str_range:Nnn</code>	2422, 2534
<code>\str_set:Nn</code>	238, 322, 710, 1851, 1903, 1905, 1907, 1909, 1911, 1914, 1916, 1921, 1934, 1947, 1949, 2035, 2036, 2053, 2187, 2319, 2362, 5262, 5265, 5790, 5792, 5794, 5796, 5798, 5800, 5802, 5804, 5837, 5840, 5892, 6003, 6020, 6060, 6062, 6064, 6066, 6069, 6072, 6075, 6077, 6346, 6350, 6905
<code>\str_set_eq:NN</code>	714, 5838
<code>\l_tmpa_str</code>	710, 711, 713, 714
<code>\strut</code>	2482, 2488
<code>\strutbox</code>	3618, 3636, 7019, 7022
<code>\SubMatrix</code>	1400, 3150, 6931, 6975, 7718, 7726, 7733, 7737, 7802
sys commands:	
<code>\sys_if_engine_xetex_p:</code>	1082
<code>\sys_if_output_dvi_p:</code>	1082
T	
<code>\tabcolsep</code>	1102, 1678, 1682
<code>\tabskip</code>	1230
<code>\tabularnote</code>	391, 414, 439, 7894, 7917
<code>\tabularnotes</code>	2487
TeX and L ^A T _E X 2 _ε commands:	
<code>\@BTnormal</code>	1203
<code>\@addamp</code>	1509, 2255
<code>\@addamp@LaTeX</code>	1509
<code>\@addtopreamble</code>	2262
<code>\@array</code>	1516, 1520
<code>\@array@array</code>	1516
<code>\@arraycr</code>	1513
<code>\@arraycr@array</code>	1513
<code>\@arstrut</code>	2287
<code>\@arstrutbox</code>	940, 941, 980, 1232, 1234, 1236, 1239, 1241, 1985, 1996, 2011, 2017, 2344, 2348
<code>\@classx</code>	1511
<code>\@classx@array</code>	1511
<code>\@currenvir</code>	6794
<code>\@depth</code>	6846, 7208, 7232
<code>\@empty</code>	2262
<code>\@finalstrut</code>	1996

\@firststampfalse	2255	\pgfsys@getposition	1364, 1369, 1375, 1443, 1451, 1454
\@gobblethree	79	\pgfsys@markposition	1067, 1075, 1128, 1215, 1363, 2666, 2687, 2705, 2736, 2762, 2801, 2835
\@halignto	1105, 1106	\pgfutil@check@rerun	106, 107
\@height	137, 152, 177, 6844, 7207, 7231	\reserved@a	139
\@ifclassloaded	60, 63, 7504, 7514	\rvtx@ifformat@geq	66
\@ifnextchar	1282, 1520	\set@color	5884, 6221
\@ifpackageloaded	32, 35, 38, 41, 44, 48, 100, 116, 203, 7507, 7517	\tikz@library@external@loaded	1545
\@mainaux	75, 101, 1715, 1716, 1717, 1722, 5584, 5585, 5591	tex commands:	
\@mkpream	1518, 2261	\tex_mkern:D	88, 90, 92, 95
\@mkpream@array	1518	\tex_the:D	217
\@preamble	2288	\textfont	2444
\@preamerr	2255	\textit	383
\@sharp	2286	\textsuperscript	384, 385
\@tabarray	1107, 1520	\the	1735, 1753, 2258, 2260, 3013
\@tabular	1517	\thetabularnote	386
\@tabular@array	1517	\tikzexternaldisable	1547
\@tempswafalse	1735, 2258	\tikzset	1385, 1549, 3139, 4993, 5195
\@tempwattrue	1734, 2257	tl commands:	
\@temptokena	216, 217, 1733, 1753, 2256, 2260	\tl_clear:N	4382, 6394, 7280
\@whiles	1735, 2258	\tl_clear_new:N	4527, 4528, 4574, 4610, 6502, 6923, 6924, 6949, 6950, 6951, 6952
\@width	137, 6847, 7209, 7233	\tl_const:Nn	51, 52, 56, 57, 526, 2857, 2902, 7159
\@xargarraycr	1514	\tl_count:n	2425, 2537
\@xargarraycr@array	1514	\tl_gclear:N	907, 1559, 1566, 1737, 2259, 3156, 3161, 4477
\@xarraycr	1512	\tl_gclear_new:N	1285, 1286, 1287, 1288, 1289, 1290, 1291, 1558
\@xarraycr@array	1512	\tl_gput_left:Nn	1090, 1770, 4776
\@xhline	140	\tl_gput_right:Nn	1090, 1595, 1782, 1833, 1873, 1887, 2115, 2136, 2144, 2157, 2163, 2169, 3048, 3062, 3070, 3167, 4144, 4221, 4387, 4396, 4408, 4413, 4424, 4435, 4465, 4753, 4764, 5237, 5349, 5388, 5390, 5465, 5868, 6138, 6148, 6164, 6176, 6187, 6199, 6211, 6707
\array@array	1515	\tl_gset:Nn	267, 1530, 1531, 1532, 1719, 1741, 1747, 2102, 2103, 2133, 2159, 3169, 4463
\bBigg@	1338, 1340	\tl_if_blank:nTF	4490, 4493, 4499, 4502, 4508, 4511, 4517, 4520, 4624, 5815
\c@MaxMatrixCols	2980, 7584	\tl_if_blank_p:n	4682, 4686, 4736, 4740, 4933, 5121, 5825, 5830
\c@tabularnote	2477, 2505	\tl_if_empty:NnTF	1154, 1567, 1576, 1669, 1687, 2476, 3135, 3136, 3163, 4385, 4433, 4621, 4704, 4705, 4906, 5094, 5883, 6162, 6184, 6400, 6559, 6579, 6838, 7048, 7201, 7225, 7282
\col@sep	1102, 1103, 1641, 1702, 2673, 2726, 2793, 2899, 2934, 3505, 3527	\tl_if_empty:nTF	164, 691, 819, 845, 861, 877, 898, 1489, 2614, 2641, 3544, 3583, 3664, 3733, 3781, 4209, 4284, 4293, 4307, 6897, 7168, 7236, 7552
\CT@arc	120, 121	\tl_if_empty_p:N	1776, 1788, 3863, 3864
\CT@arc@	119, 124, 138, 151, 176, 190, 192, 1544, 2499, 3176, 4960, 4978, 5146, 5180, 6403, 6528, 6742, 7050	\tl_if_empty_p:n	2467, 5856
\CT@drs	126, 127	\tl_if_eq:NnTF	1156, 1739, 2230, 2381, 2406, 2525, 5012, 5167, 5177, 5214, 6954, 6957, 6961, 6964, 7065, 7082
\CT@drsc@	130, 196, 198, 4932, 4933, 4937, 5120, 5121, 5125	\tl_if_eq:nnTF	424, 703, 828, 2130, 2152, 4459
\CT@everycr	1223	\tl_if_eq_p:NN	3826
\CT@row@color	1225	\tl_if_exist:NnTF	1561
\endarray@array	1519	\tl_if_in:NnTF	4590, 4678, 4701, 4732, 5292
\if@firststamp	2255		
\if@tempswa	1735, 2258		
\insert@column	1510		
\insert@column@array	1510		
\NC@	1175, 3014		
\NC@do	3013		
\NC@find	218		
\NC@find@V	1732		
\NC@list	1735, 2258, 3013		
\NC@rewrite@S	214		
\new@ifnextchar	1282		
\newcol@	1177, 1178, 3016, 3017		
\nicematrix@redefine@check@rerun	101, 104		
\pgf@relevantforpicturesizefalse	1366, 1481, 3213, 3824, 3997, 4472, 4586, 4915, 4970, 4986, 5103, 5156, 5188, 5674, 5684, 5695, 6225, 6399, 6527, 6729, 6804, 6981, 7285, 7438		

5.4	The command <code>\diagbox</code>	11
5.5	Commands for customized rules	11
6	The color of the rows and columns	13
6.1	Use of <code>colortbl</code>	13
6.2	The tools of <code>nicematrix</code> in the <code>\CodeBefore</code>	14
6.3	Color tools with the syntax of <code>colortbl</code>	18
7	The command <code>\RowStyle</code>	18
8	The width of the columns	19
8.1	Basic tools	19
8.2	The columns <code>V</code> of <code>varwidth</code>	20
8.3	The columns <code>X</code>	21
9	The exterior rows and columns	21
10	The continuous dotted lines	23
10.1	The option <code>nullify-dots</code>	24
10.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	25
10.3	How to generate the continuous dotted lines transparently	26
10.4	The labels of the dotted lines	26
10.5	Customisation of the dotted lines	27
10.6	The dotted lines and the rules	28
11	The <code>\CodeAfter</code>	28
11.1	The command <code>\line</code> in the <code>\CodeAfter</code>	29
11.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	29
11.3	The commands <code>\OverBrace</code> and <code>\UnderBrace</code> in the <code>\CodeAfter</code>	31
12	The notes in the tabulars	32
12.1	The footnotes	32
12.2	The notes of tabular	32
12.3	Customisation of the tabular notes	34
12.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	36
13	Other features	36
14	Autres fonctionnalités	36
14.1	Command <code>\ShowCellNames</code>	36
14.2	Use of the column type <code>S</code> of <code>siunitx</code>	36
14.3	Alignment option in <code>{NiceMatrix}</code>	37
14.4	The command <code>\rotate</code>	37
14.5	The option <code>small</code>	37
14.6	The counters <code>iRow</code> and <code>jCol</code>	38
14.7	The option <code>light-syntax</code>	39
14.8	Color of the delimiters	39
14.9	The environment <code>{NiceArrayWithDelims}</code>	39
14.10	The command <code>\OnlyMainNiceMatrix</code>	40
15	Use of Tikz with <code>nicematrix</code>	40
15.1	The nodes corresponding to the contents of the cells	40
15.1.1	The columns <code>V</code> of <code>varwidth</code>	41
15.2	The medium nodes and the large nodes	41
15.3	The nodes which indicate the position of the rules	43
15.4	The nodes corresponding to the command <code>\SubMatrix</code>	44
16	API for the developpers	44

17	Technical remarks	45
17.1	Diagonal lines	45
17.2	The empty cells	46
17.3	The option exterior-arraycolsep	46
17.4	Incompatibilities	47
18	Examples	47
18.1	Utilisation of the key 'tikz' of the command <code>\Block</code>	47
18.2	Notes in the tabulars	48
18.3	Dotted lines	49
18.4	Dotted lines which are no longer dotted	50
18.5	Dashed rules	51
18.6	Stacks of matrices	51
18.7	How to highlight cells of a matrix	55
18.8	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	57
19	Implementation	58
20	History	241
	Index	249