

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

March 2, 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}`.

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

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

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

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

*This document corresponds to the version 6.7 of `nicematrix`, at the date of 2022/03/02.

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

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

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

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

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

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

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

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

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

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;
- the key `fill` takes in as value a color and fills the block with that color;
- the key `draw` takes in as value a color and strokes the frame of the block with that color (the default value of that key is the current color of the rules of the array);
- the key `color` takes in as value a color and apply that color the content of the block but draws also the frame of the block with that color;
- the key `line-width` is the width (thickness) of the frame (this key should be used only when the key `draw` or the key `hvlines` is in force);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt⁸);
- the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`);
- the keys `hlines`, `vlines` and `hvlines` draw all the corresponding rules in the block;
- when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz⁹ by using as options the value of that key `tikz` (which must be a list of keys allowed for a Tikz path). For examples, cf. p. 46;
- 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);
- **New 6.5** 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`;
- **Nouveau 6.7** 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. 50).

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`.

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. 45).

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

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

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

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

5.5 Dotted rules

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

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

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

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

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

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

It's possible to change in `nicematrix` the letter used to specify a vertical dotted line with the option `letter-for-dotted-lines` available in `\NiceMatrixOptions`.

Remark: In the package `array` (on which the package `nicematrix` relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule¹⁶. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

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

¹⁶In fact, with `array`, this is true only for `\hline` and “|” but not for `\cline`: cf p. 8

5.6 Commands for customized rules

New 6.5 It's 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 letter¹⁷ that the user will use in the preamble of an environment with preamble (such as `{NiceTabular}`) in order to specify a vertical rule.

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

- *First possibility*

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

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

- *Second possibility*

The key `dotted` forces a style with dotted rules such as those created by `\hdottedline` or the letter “:” in the preamble (cf. p. 11). The key `color` may be used also in that case.

- **New 6.6** *Third 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 possible to specify a reservation (horizontal for a vertical rule and vertical for an horizontal one) with the key `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`).

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 as do all 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}
```

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

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

Here is an example of the key `tikz`.

```
\documentclass{article}
\usepackage{nicematrix,tikz}
\usetikzlibrary{decorations.pathmorphing}

\NiceMatrixOptions
{
  custom-line =
  {
    letter = I ,
    tikz = { decorate, decoration = { coil, aspect = 0 } } ,
    width = 2 mm
  }
}

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

one	⋈	two	⋈	three
four	⋈	five	⋈	six
seven	⋈	eight	⋈	nine

6 The color of the rows and columns

6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

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

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

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

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

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

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

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

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

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

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. 42.

```

\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. 36).

- 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.
- 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 mode and text mode.

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

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

The command `\rotate` is described p. 36.

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

New 6.3

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

That point is illustrated on the following examples.

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

- first item
- second item

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

- first item
- second item

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

When the package `varwidth` is loaded, the columns `V` of `varwidth` are supported by `nicematrix`. Concerning `nicematrix`, one of the interests of this type of columns is that, for a cell of a column of type `V`, the PGF/Tikz node created by `nicematrix` for the content of that cell has a width adjusted to the content of the cell : cf. p. 40. 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[1]}[hvlines]
a rather long text which fits on several lines
& a rather long text which fits on several lines \\
a shorter text & a shorter text
\end{NiceTabular}
```

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

9 The exterior rows and columns

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

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

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. 38) 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. 12).
- 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. 45).

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).

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

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

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

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

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

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

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

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

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

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

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`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. 28) 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`. There is also a key `transparent` which is an alias for the conjunction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.

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

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

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

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

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

The option `xdots/color`

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

The option `xdots/shorten`

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

The option `xdots/line-style`

It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).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 (and this system uses PGF and not Tikz). This style is called `standard` and that’s the initial value of the parameter `xdots/line-style`.

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

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

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

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.

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

10.6 The dotted lines and the rules

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

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

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

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. 39.

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

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

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

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

³⁷On the other side, the command `\line` in the `\CodeAfter` (cf. p. 28) does *not* create block.

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

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. 44).

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

$$\left[\begin{array}{cccccc} 1 & \cdots & 1 & 2 & \cdots & 2 \\ 0 & \ddots & & \vdots & \vdots & \hspace*{2.5cm} & \vdots \\ \vdots & \ddots & & & & & \\ 0 & \cdots & 0 & 1 & 2 & \cdots & 2 \end{array} \right]$$

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

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.

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

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

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:

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}]`).

- `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 $\backslash ht$ + depth $\backslash dp$);
- `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.

```


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


```

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

```


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


```

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. 43.

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.

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

```

 $\begin{pNiceArray}{(c)(c)(c)}$ 
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`

New 6.4

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}
```

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

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:

- `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) ;
- **New 6.7** `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}
```

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

12 The notes in the tabulars

12.1 The footnotes

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

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

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

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

12.2 The notes of tabular

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

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

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

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

^a Achard is an old family of the Poitou.

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

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

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

For an illustration of some of those remarks, see table 1, p. 33. This table has been composed with the following code.

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

Table 1: Use of `\tabularnote`^a

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

Some text before the notes.

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

^b Considered as the first nurse of history.

^c Nicknamed “the Lady with the Lamp”.

^d The label of the note is overlapping.

12.3 Customisation of the tabular notes

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

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

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

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

We detail these keys.

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

Initial value: `false`

That key is also available within a given environment.

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

Initial value: `false`

That key is also available within a given environment.

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

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

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

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

Initial value: `#1`

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

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

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

Initial value: `#1`

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

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

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

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

```
noitemsep , leftmargin = * , align = left , labelsep = Opt
```

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. 33).

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`).

For an example of customisation of the tabular notes, see p. 47.

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

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

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

13 Other features

13.1 Use of the column type S of siunitx

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

```


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


```

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

13.2 Alignment option in `{NiceMatrix}`

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

```


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


```

13.3 The command `\rotate`

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

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

```

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

```

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

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

```

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

```

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

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

13.4 The option `small`

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

```

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

```

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

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

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

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

13.5 The counters `iRow` and `jCol`

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current 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).

```

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

```

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

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

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.

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

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

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

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

13.6 The option `light-syntax`

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

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

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

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

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

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

13.7 Color of the delimiters

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

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

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

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

13.8 The environment `\NiceArrayWithDelims`

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

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

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

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.

13.9 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

14 Use of Tikz with `nicematrix`

14.1 The nodes corresponding to the contents of the cells

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

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

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

```


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


```

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

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

```


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


```

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 54).

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`.

14.1.1 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. 42).

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

In fact, the package `nicematrix` can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option `create-medium-nodes` and the second ones with the option `create-large-nodes`.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 (with-out use of `margin` nor `extra-margin`).

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

The nodes we have described are not available by default in the `\CodeBefore` (described p. 14). It’s possible to have these nodes available in the `\CodeBefore` by using the key `create-cell-nodes` of the keyword `\CodeBefore` (in that case, the nodes are created first before the construction of the array by using informations written on the `aux` file and created a second time during the contruction of the array itself).

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

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

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}

```

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

14.3 The nodes which indicate the position of the rules

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

$\bullet^{1.5}$	\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}$

```

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

14.4 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}\}`).

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

15 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.

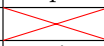
Example : We want to write a command `\crossbox` to draw a cross in the current cell. This command will take in an optional argument between square brackets for a list of pairs *key-value* which will be given to Tikz before the drawing.
It's possible to program such command `\crossbox` as follows, explicitly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

16 Technical remarks

16.1 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`.

```

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

```

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

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

The same example without parallelization:

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

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

16.2 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.

16.3 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).

16.4 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. 12.

17 Examples

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

The key `tikz` of the command `\Block` is available only when Tikz is loaded.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>	

17.2 Notes in the tabulars

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

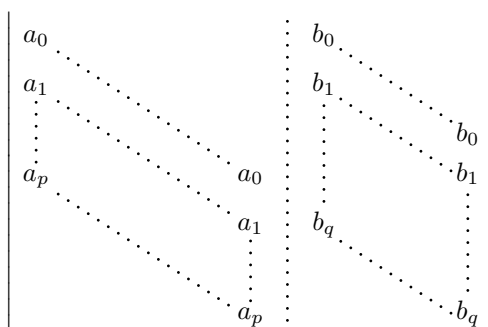
17.3 Dotted lines

An example with the resultant of two polynoms:

```

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

```



An example for a linear system:

```

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

```


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

17.4 Dotted lines which are no longer dotted

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

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

$$\left(\begin{array}{cccc} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{array} \right) \begin{array}{l} \leftarrow i \\ \leftarrow j \end{array}$$

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

```

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

17.5 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}

```

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

17.6 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}\]

```

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

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

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

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

17.7 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”.

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

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

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

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

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

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

```
\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}\}

```

18 Implementation

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

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

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

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

Declaration of the package and packages loaded

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

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

<@@=nicematrix>

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

```

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

```

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

```

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

```

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

We load some packages. The package `xparse` is still loaded for use on Overleaf. However, since oct. 2021, Overleaf uses TeXLive 2021 and we will be able to delete that row.

```

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

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

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

```

Technical definitions

```

21 \bool_new:N \c_@@_arydshln_loaded_bool
22 \bool_new:N \c_@@_booktabs_loaded_bool
23 \bool_new:N \c_@@_enumitem_loaded_bool
24 \bool_new:N \c_@@_tabularx_loaded_bool
25 \bool_new:N \c_@@_tikz_loaded_bool
26 \bool_new:N \c_@@_varwidth_loaded_bool
27 \hook_gput_code:nnn { begindocument } { . }
28 {
29   \@ifpackageloaded { varwidth }
30   { \bool_set_true:N \c_@@_varwidth_loaded_bool }
31   { }
32   \@ifpackageloaded { arydshln }
33   { \bool_set_true:N \c_@@_arydshln_loaded_bool }
34   { }
35   \@ifpackageloaded { booktabs }
36   { \bool_set_true:N \c_@@_booktabs_loaded_bool }
37   { }
38   \@ifpackageloaded { enumitem }
39   { \bool_set_true:N \c_@@_enumitem_loaded_bool }
40   { }
41   \@ifpackageloaded { tabularx }
42   { \bool_set_true:N \c_@@_tabularx_loaded_bool }
43   { }
44   \@ifpackageloaded { tikz }
45   {

```

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.

```

46   \bool_set_true:N \c_@@_tikz_loaded_bool
47   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
48   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
49 }
50 {
51   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
52   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }

```

```

53   }
54 }

```

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

```

55 \bool_new:N \c_@@_revtex_bool
56 \ifclassloaded { revtex4-1 }
57   { \bool_set_true:N \c_@@_revtex_bool }
58   { }
59 \ifclassloaded { revtex4-2 }
60   { \bool_set_true:N \c_@@_revtex_bool }
61   { }

```

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

```

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

```

```

63 \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.

```

64 \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.

```

65 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
66   {
67     \iow_now:Nn \@mainaux
68     {
69       \ExplSyntaxOn
70       \cs_if_free:NT \pgfsyspdfmark
71         { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
72       \ExplSyntaxOff
73     }
74     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
75   }

```

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.

```

76 \ProvideDocumentCommand \iddots { }
77 {
78   \mathinner
79   {
80     \tex_mkern:D 1 mu
81     \box_move_up:nn { 1 pt } { \hbox:n { . } }
82     \tex_mkern:D 2 mu
83     \box_move_up:nn { 4 pt } { \hbox:n { . } }
84     \tex_mkern:D 2 mu
85     \box_move_up:nn { 7 pt }
86     { \vbox:n { \kern 7 pt \hbox:n { . } } }
87     \tex_mkern:D 1 mu
88   }
89 }

```

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.

```

90 \hook_gput_code:nnn { begindocument } { . }
91 {
92   \@ifpackageloaded { booktabs }
93     { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
94     { }
95 }
96 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
97 {
98   \cs_set_eq:NN \@_old_pgfutil@check@rerun \pgfutil@check@rerun

```

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

```

99   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
100   {
101     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
102     { \@_old_pgfutil@check@rerun { ##1 } { ##2 } }
103   }
104 }

```

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

```

105 \bool_new:N \c_@@_colortbl_loaded_bool
106 \hook_gput_code:nnn { begindocument } { . }
107 {
108   \@ifpackageloaded { colortbl }
109     { \bool_set_true:N \c_@@_colortbl_loaded_bool }
110     {

```

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.

```

111   \cs_set_protected:Npn \CT@arc@ { }
112   \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
113   \cs_set:Npn \CT@arc@ #1 #2
114   {
115     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
116       { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
117   }

```

Idem for `\CT@drs@`.

```

118   \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
119   \cs_set:Npn \CT@drs@ #1 #2
120   {
121     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
122       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
123   }
124   \cs_set:Npn \hline
125   {
126     \noalign { \ifnum 0 = ` } \fi
127     \cs_set_eq:NN \hskip \vskip
128     \cs_set_eq:NN \vrule \hrule
129     \cs_set_eq:NN \@width \@height
130     { \CT@arc@ \vline }
131     \futurelet \reserved@a
132     \@xhline
133   }
134 }
135 }

```

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.

```

136 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
137 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
138 {

```

```

139 \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
140 \int_compare:nNnT { #1 } > 1 { \multispan { \int_eval:n { #1 - 1 } } & }
141 \multispan { \int_eval:n { #2 - #1 + 1 } }
142 {
143   \CT@arc@
144   \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

145   \skip_horizontal:N \c_zero_dim
146 }

```

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`.

```

147 \everycr { }
148 \cr
149 \noalign { \skip_vertical:N -\arrayrulewidth }
150 }

```

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.

```

151 \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`.

```

152 { \@@_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*.

```

153 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
154 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
155 {
156   \tl_if_empty:nTF { #3 }
157   { \@@_cline_iii:w #1|#2-#2 \q_stop }
158   { \@@_cline_ii:w #1|#2-#3 \q_stop }
159 }
160 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
161 { \@@_cline_iii:w #1|#2-#3 \q_stop }
162 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
163 {

```

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).

```

164 \int_compare:nNnT { #1 } < { #2 }
165 { \multispan { \int_eval:n { #2 - #1 } } & }
166 \multispan { \int_eval:n { #3 - #2 + 1 } }
167 {
168   \CT@arc@
169   \leaders \hrule \@height \arrayrulewidth \hfill
170   \skip_horizontal:N \c_zero_dim
171 }

```

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

```

172 \peek_meaning_remove_ignore_spaces:NTF \cline
173 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
174 { \everycr { } \cr }
175 }
176 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

The following command is a small shortcut.

```

177 \cs_new:Npn \@@_math_toggle_token:

```

⁵⁹See question 99041 on TeX StackExchange.

```

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

179 \cs_new_protected:Npn \@@_set_CT@arc@:
180 { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
181 \cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
182 { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
183 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
184 { \cs_set:Npn \CT@arc@ { \color { #1 } } }

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

191 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

```

The column S of siunitx

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

```

192 \bool_new:N \c_@@_siunitx_loaded_bool
193 \hook_gput_code:nnn { begindocument } { . }
194 {
195   \@ifpackageloaded { siunitx }
196   { \bool_set_true:N \c_@@_siunitx_loaded_bool }
197   { }
198 }

```

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

```

199 \hook_gput_code:nnn { begindocument } { . }
200 {
201   \bool_if:nTF { ! \c_@@_siunitx_loaded_bool }
202   { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
203   {
204     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
205     {
206       \renewcommand*{\NC@rewrite@S}[1] []
207       {

```

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

```

208       \@temptokena \exp_after:wN
209       { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
210       \NC@find
211     }
212   }
213 }
214 }

```

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.

```

215 \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).

```

216 \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.

```
217 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
218 { \int_use:N \g_@@_env_int }
```

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

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

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

```
221 \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).

```
222 \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.).

```
223 \dim_new:N \l_@@_col_width_dim
224 \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.

```
225 \int_new:N \g_@@_row_total_int
226 \int_new:N \g_@@_col_total_int
```

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

```
227 \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.

```
228 \str_new:N \l_@@_hpos_cell_str
229 \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`.

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

Idem pour the mono-row blocks.

```
231 \dim_new:N \g_@@_blocks_ht_dim
232 \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}`).

```
233 \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`.

```
234 \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.

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

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

```
236 \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.

```
237 \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.

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

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

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

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

```
240 \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.

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

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

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

```
243 \cs_new_protected:Npn \@@_test_if_math_mode:
244 {
245   \if_mode_math: \else:
246     \@@_fatal:n { Outside-math-mode }
247   \fi:
248 }
```

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

```
249 \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.

```
250 \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”.

```
251 \colorlet { nicematrix-last-col } { . }
252 \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*).

```
253 \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*.

```
254 \tl_set:Nn \g_@@_com_or_env_str { environment }
```

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

```
255 \cs_new:Npn \@@_full_name_env:
256 {
257   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
258   { command \space \c_backslash_str \g_@@_name_env_str }
259   { environment \space \{ \g_@@_name_env_str \} }
260 }
```

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*.

```
261 \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.

```
262 \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.

```
263 \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.

```
264 \int_new:N \l_@@_old_iRow_int
265 \int_new:N \l_@@_old_jCol_int
```

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

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

```
266 \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.

```
267 \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.

```
268 \bool_new:N \l_@@_X_columns_aux_bool
269 \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.

```
270 \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).

```
271 \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).

```
272 \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`).

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

```
274 \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).

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

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

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

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

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

```
279 \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.

```
280 \dim_zero_new:N \l_@@_tmpc_dim
```

```
281 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
282 \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”.

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

```
284 \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.

```
285 \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!).

```
286 \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: $\{imin\}\{jmin\}\{imax\}\{jmax\}\{name\}$.

```
287 \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.

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

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

```
289 \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`).

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

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

```
291 \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).

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

```
293 \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`).

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

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

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

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

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

```
298 \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).

```
299 \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`.

```
300 \tl_new:N \l_@@_fill_tl
301 \tl_new:N \l_@@_draw_tl
302 \seq_new:N \l_@@_tikz_seq
303 \clist_new:N \l_@@_borders_clist
304 \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`.

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

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

```
306 \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`).

```
307 \str_new:N \l_@@_hpos_block_str
308 \str_set:Nn \l_@@_hpos_block_str { c }
309 \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`.

```
310 \tl_new:N \l_@@_vpos_of_block_tl
311 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

```
312 \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).

```
313 \bool_new:N \l_@@_vlines_block_bool
314 \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.

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

316 \dim_new:N \l_@@_submatrix_extra_height_dim
317 \dim_new:N \l_@@_submatrix_left_xshift_dim
318 \dim_new:N \l_@@_submatrix_right_xshift_dim
319 \clist_new:N \l_@@_hlines_clist
320 \clist_new:N \l_@@_vlines_clist
321 \clist_new:N \l_@@_submatrix_hlines_clist
322 \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.

```
323 \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.

```
324 \int_new:N \l_@@_first_row_int
325 \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.

```
326 \int_new:N \l_@@_first_col_int
327 \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).

```
328 \int_new:N \l_@@_last_row_int
329 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
331 \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.

```
332 \int_new:N \l_@@_last_col_int
333 \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}
```

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.

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.

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

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

Some utilities

```
335 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
336 {
337   \tl_set:Nn \l_tmpa_tl { #1 }
338   \tl_set:Nn \l_tmpb_tl { #2 }
339 }
```

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.

```
340 \cs_new_protected:Npn \@@_expand_clist:N #1
341 {
342   \clist_if_in:NnF #1 { all }
343   {
344     \clist_clear:N \l_tmpa_clist
345     \clist_map_inline:Nn #1
346     {
347       \tl_if_in:nnTF { ##1 } { - }
348       { \@@_cut_on_hyphen:w ##1 \q_stop }
349       {
350         \tl_set:Nn \l_tmpa_tl { ##1 }
351         \tl_set:Nn \l_tmpb_tl { ##1 }
352       }
353       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
354       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
355     }
356     \tl_set_eq:NN #1 \l_tmpa_clist
357   }
358 }
```

The command `\tablarnote`

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

```
359 \newcounter { tablarnote }
```

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

```
360 \seq_new:N \g_@@_tablarnotes_seq
```

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

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

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

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

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

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

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

```
364 \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`.

```
365 \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`.

```
366 \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`).

```
367 \hook_gput_code:nnn { begindocument } { . }
368 {
369   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
370   {
371     \NewDocumentCommand \tabularnote { m }
372     { \@@_error:n { enumitem-not-loaded } }
373   }
374   {
```

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.

```
375   \newlist { tabularnotes } { enumerate } { 1 }
376   \setlist [ tabularnotes ]
377   {
378     topsep = 0pt ,
379     noitemsep ,
380     leftmargin = * ,
381     align = left ,
382     labelsep = 0pt ,
383     label =
384       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
385   }
386   \newlist { tabularnotes* } { enumerate* } { 1 }
387   \setlist [ tabularnotes* ]
388   {
389     afterlabel = \nobreak ,
390     itemjoin = \quad ,
391     label =
392       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
393   }
```

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

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

```

394 \NewDocumentCommand \tabularnote { m }
395 {
396   \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
397   { \@@_error:n { tabularnote~forbidden } }
398   {

```

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

```

399   \int_incr:N \l_@@_number_of_notes_int

```

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

```

400   \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
401   \peek_meaning:NF \tabularnote
402   {

```

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.

```

403   \hbox_set:Nn \l_tmpa_box
404   {

```

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

```

405   \@@_notes_label_in_tabular:n
406   {
407     \stepcounter { tabularnote }
408     \@@_notes_style:n { tabularnote }
409     \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
410     {
411       ,
412       \stepcounter { tabularnote }
413       \@@_notes_style:n { tabularnote }
414     }
415   }
416 }

```

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.

```

417   \addtocounter { tabularnote } { -1 }
418   \refstepcounter { tabularnote }
419   \int_zero:N \l_@@_number_of_notes_int
420   \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).

```

421   \skip_horizontal:n { \box_wd:N \l_tmpa_box }
422   }
423 }
424 }
425 }
426 }

```

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.

```

427 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
428 {

```



```

429 \begin { pgfscope }
430 \pgfset
431 {
432     outer-sep = \c_zero_dim ,
433     inner-sep = \c_zero_dim ,
434     minimum-size = \c_zero_dim
435 }
436 \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
437 \pgfnode
438 { rectangle }
439 { center }
440 {
441     \vbox_to_ht:nn
442     { \dim_abs:n { #5 - #3 } }
443     {
444         \vfill
445         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
446     }
447 }
448 { #1 }
449 { }
450 \end { pgfscope }
451 }

```

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.

```

452 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
453 {
454     \begin { pgfscope }
455     \pgfset
456     {
457         outer-sep = \c_zero_dim ,
458         inner-sep = \c_zero_dim ,
459         minimum-size = \c_zero_dim
460     }
461     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
462     \pgfpointdiff { #3 } { #2 }
463     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
464     \pgfnode
465     { rectangle }
466     { center }
467     {
468         \vbox_to_ht:nn
469         { \dim_abs:n \l_tmpb_dim }
470         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
471     }
472     { #1 }
473     { }
474     \end { pgfscope }
475 }

```

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.

```

476 \bool_new:N \l_@@_colortbl-like_bool

```

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

```
477 \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`).

```
478 \dim_new:N \l_@@_cell_space_top_limit_dim
479 \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.

```
480 \dim_new:N \l_@@_inter_dots_dim
481 \hook_gput_code:nnn { begindocument } { . }
482 { \dim_set:Nn \l_@@_inter_dots_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).

```
483 \dim_new:N \l_@@_xdots_shorten_dim
484 \hook_gput_code:nnn { begindocument } { . }
485 { \dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 em } }
```

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.

```
486 \dim_new:N \l_@@_radius_dim
487 \hook_gput_code:nnn { begindocument } { . }
488 { \dim_set:Nn \l_@@_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.

```
489 \tl_new:N \l_@@_xdots_line_style_tl
490 \tl_const:Nn \c_@@_standard_tl { standard }
491 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
492 \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).

```
493 \tl_new:N \l_@@_baseline_tl
494 \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`).

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

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

```
496 \bool_new:N \l_@@_parallelize_diags_bool
497 \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.

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

```
499 \dim_new:N \l_@@_notes_above_space_dim
500 \hook_gput_code:nnn { begindocument } { . }
501 { \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.).

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

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

```
503 \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).

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

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

```
505 \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.

```
506 \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”.

```
507 \bool_new:N \l_@@_medium_nodes_bool
508 \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).

```
509 \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`).

```
510 \dim_new:N \l_@@_left_margin_dim
511 \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`.

```
512 \dim_new:N \l_@@_extra_left_margin_dim
513 \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.

```
514 \tl_new:N \l_@@_end_of_row_tl
515 \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 “:”.

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

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

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

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

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

```
519 \keys_define:nn { NiceMatrix / xdots }
520 {
521   line-style .code:n =
522   {
523     \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.

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

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

```
535   shorten .value_required:n = true ,
```

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

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

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

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

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

```

548 \keys_define:nn { NiceMatrix / Global }
549 {
550   custom-line .code:n = \l_@@_custom_line:n { #1 } ,
551   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
552   delimiters .value_required:n = true ,
553   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
554   rules .value_required:n = true ,
555   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
556   standard-cline .default:n = true ,
557   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
558   cell-space-top-limit .value_required:n = true ,
559   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
560   cell-space-bottom-limit .value_required:n = true ,
561   cell-space-limits .meta:n =
562   {
563     cell-space-top-limit = #1 ,
564     cell-space-bottom-limit = #1 ,
565   } ,
566   cell-space-limits .value_required:n = true ,
567   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
568   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
569   light-syntax .default:n = true ,
570   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
571   end-of-row .value_required:n = true ,
572   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
573   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
574   last-row .int_set:N = \l_@@_last_row_int ,
575   last-row .default:n = -1 ,
576   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
577   code-for-first-col .value_required:n = true ,
578   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
579   code-for-last-col .value_required:n = true ,
580   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
581   code-for-first-row .value_required:n = true ,
582   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
583   code-for-last-row .value_required:n = true ,
584   hlines .clist_set:N = \l_@@_hlines_clist ,
585   vlines .clist_set:N = \l_@@_vlines_clist ,
586   hlines .default:n = all ,
587   vlines .default:n = all ,
588   vlines-in-sub-matrix .code:n =
589   {
590     \tl_if_single_token:nTF { #1 }
591     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
592     { \@@_error:n { One-letter-allowed } }
593   } ,
594   vlines-in-sub-matrix .value_required:n = true ,
595   hvlines .code:n =
596   {
597     \clist_set:Nn \l_@@_vlines_clist { all }
598     \clist_set:Nn \l_@@_hlines_clist { all }
599   } ,
600   hvlines-except-borders .code:n =
601   {
602     \clist_set:Nn \l_@@_vlines_clist { all }
603     \clist_set:Nn \l_@@_hlines_clist { all }
604     \bool_set_true:N \l_@@_except_borders_bool
605   } ,
606   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.

```

607   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
608   renew-dots .value_forbidden:n = true ,
609   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
610   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
611   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
612   create-extra-nodes .meta:n =
613     { create-medium-nodes , create-large-nodes } ,
614   left-margin .dim_set:N = \l_@@_left_margin_dim ,
615   left-margin .default:n = \arraycolsep ,
616   right-margin .dim_set:N = \l_@@_right_margin_dim ,
617   right-margin .default:n = \arraycolsep ,
618   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
619   margin .default:n = \arraycolsep ,
620   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
621   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
622   extra-margin .meta:n =
623     { extra-left-margin = #1 , extra-right-margin = #1 } ,
624   extra-margin .value_required:n = true ,
625   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
626   respect-arraystretch .default:n = true
627 }

```

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

```

628 \keys_define:nn { NiceMatrix / Env }
629 {

```

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

```

630   hvlines-except-corners .code:n =
631   {
632     \@@_error:n { hvlines-except-corners }
633     \group_begin:
634     \globaldefs = 1
635     \@@_msg_redirect_name:nn { hvlines-except-corners } { none }
636     \group_end:
637     \clist_set:Nn \l_@@_corners_clist { #1 }
638     \clist_set:Nn \l_@@_vlines_clist { all }
639     \clist_set:Nn \l_@@_hlines_clist { all }
640   } ,
641   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
642   corners .clist_set:N = \l_@@_corners_clist ,
643   corners .default:n = { NW , SW , NE , SE } ,
644   code-before .code:n =
645   {
646     \tl_if_empty:nF { #1 }
647     {
648       \tl_put_right:Nn \l_@@_code_before_tl { #1 }
649       \bool_set_true:N \l_@@_code_before_bool
650     }
651   } ,

```

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

```

652   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
653   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
654   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
655   baseline .tl_set:N = \l_@@_baseline_tl ,
656   baseline .value_required:n = true ,
657   columns-width .code:n =
658     \tl_if_eq:nnTF { #1 } { auto }
659     { \bool_set_true:N \l_@@_auto_columns_width_bool }

```

```

660     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
661     columns-width .value_required:n = true ,
662     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.

```

663     \legacy_if:nF { measuring@ }
664     {
665         \str_set:Nn \l_tmpa_str { #1 }
666         \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
667         { \@@_error:nn { Duplicate-name } { #1 } }
668         { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
669         \str_set_eq:NN \l_@@_name_str \l_tmpa_str
670     } ,
671     name .value_required:n = true ,
672     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
673     code-after .value_required:n = true ,
674     colortbl-like .code:n =
675         \bool_set_true:N \l_@@_colortbl_like_bool
676         \bool_set_true:N \l_@@_code_before_bool ,
677     colortbl-like .value_forbidden:n = true
678 }

679 \keys_define:nn { NiceMatrix / notes }
680 {
681     para .bool_set:N = \l_@@_notes_para_bool ,
682     para .default:n = true ,
683     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
684     code-before .value_required:n = true ,
685     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
686     code-after .value_required:n = true ,
687     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
688     bottomrule .default:n = true ,
689     style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
690     style .value_required:n = true ,
691     label-in-tabular .code:n =
692         \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
693     label-in-tabular .value_required:n = true ,
694     label-in-list .code:n =
695         \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
696     label-in-list .value_required:n = true ,
697     enumitem-keys .code:n =
698         {
699             \hook_gput_code:nnn { begindocument } { . }
700             {
701                 \bool_if:NT \c_@@_enumitem_loaded_bool
702                 { \setlist* [ tabularnotes ] { #1 } }
703             }
704         } ,
705     enumitem-keys .value_required:n = true ,
706     enumitem-keys-para .code:n =
707         {
708             \hook_gput_code:nnn { begindocument } { . }
709             {
710                 \bool_if:NT \c_@@_enumitem_loaded_bool
711                 { \setlist* [ tabularnotes* ] { #1 } }
712             }
713         } ,
714     enumitem-keys-para .value_required:n = true ,
715     unknown .code:n = \@@_error:n { Unknown-key-for-notes }
716 }

717 \keys_define:nn { NiceMatrix / delimiters }
718 {
719     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,

```

```

720 max-width .default:n = true ,
721 color .tl_set:N = \l_@@_delimiters_color_tl ,
722 color .value_required:n = true ,
723 }

```

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

```

724 \keys_define:nn { NiceMatrix }
725 {
726   NiceMatrixOptions .inherit:n =
727     { NiceMatrix / Global } ,
728   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
729   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
730   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
731   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
732   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
733   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
734   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
735   NiceMatrix .inherit:n =
736     {
737       NiceMatrix / Global ,
738       NiceMatrix / Env ,
739     } ,
740   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
741   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
742   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
743   NiceTabular .inherit:n =
744     {
745       NiceMatrix / Global ,
746       NiceMatrix / Env
747     } ,
748   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
749   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
750   NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
751   NiceArray .inherit:n =
752     {
753       NiceMatrix / Global ,
754       NiceMatrix / Env ,
755     } ,
756   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
757   NiceArray / rules .inherit:n = NiceMatrix / rules ,
758   NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
759   pNiceArray .inherit:n =
760     {
761       NiceMatrix / Global ,
762       NiceMatrix / Env ,
763     } ,
764   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
765   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
766   pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
767 }

```

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

```

768 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
769 {
770   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
771   width .value_required:n = true ,
772   last-col .code:n = \tl_if_empty:nF { #1 }
773     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
774     \int_zero:N \l_@@_last_col_int ,
775   small .bool_set:N = \l_@@_small_bool ,
776   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.

```
777   renew-matrix .code:n = \@@_renew_matrix: ,
778   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}`.

```
779   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.

```
780   columns-width .code:n =
781     \tl_if_eq:nnTF { #1 } { auto }
782     { \@@_error:n { Option-auto-for-columns-width } }
783     { \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.

```
784   allow-duplicate-names .code:n =
785     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
786   allow-duplicate-names .value_forbidden:n = true ,
```

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

```
787   letter-for-dotted-lines .code:n =
788     {
789       \tl_if_single_token:nTF { #1 }
790       { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
791       { \@@_error:n { One-letter-allowed } }
792     } ,
793   letter-for-dotted-lines .value_required:n = true ,
794   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
795   notes .value_required:n = true ,
796   sub-matrix .code:n =
797     \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
798   sub-matrix .value_required:n = true ,
799   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
800 }

801 \str_new:N \l_@@_letter_for_dotted_lines_str
802 \str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

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

```
803 \NewDocumentCommand \NiceMatrixOptions { m }
804 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }
```

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

```
805 \keys_define:nn { NiceMatrix / NiceMatrix }
806 {
807   last-col .code:n = \tl_if_empty:nTF {#1}
808     {
809       \bool_set_true:N \l_@@_last_col_without_value_bool
810       \int_set:Nn \l_@@_last_col_int { -1 }
811     }
```

```

812         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
813     l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
814     r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
815     small .bool_set:N = \l_@@_small_bool ,
816     small .value_forbidden:n = true ,
817     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
818 }

```

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

```

819 \keys_define:nn { NiceMatrix / NiceArray }
820 {

```

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.

```

821     small .bool_set:N = \l_@@_small_bool ,
822     small .value_forbidden:n = true ,
823     last-col .code:n = \tl_if_empty:nF { #1 }
824         { \@@_error:n { last-col-non-empty-for-NiceArray } }
825         \int_zero:N \l_@@_last_col_int ,
826     notes / para .bool_set:N = \l_@@_notes_para_bool ,
827     notes / para .default:n = true ,
828     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
829     notes / bottomrule .default:n = true ,
830     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
831     tabularnote .value_required:n = true ,
832     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
833     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
834     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
835 }

836 \keys_define:nn { NiceMatrix / pNiceArray }
837 {
838     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
839     last-col .code:n = \tl_if_empty:nF { #1 }
840         { \@@_error:n { last-col-non-empty-for-NiceArray } }
841         \int_zero:N \l_@@_last_col_int ,
842     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
843     small .bool_set:N = \l_@@_small_bool ,
844     small .value_forbidden:n = true ,
845     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
846     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
847     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
848 }

```

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

```

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

```

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.

```

851     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
852         \bool_set_true:N \l_@@_width_used_bool ,
853     width .value_required:n = true ,
854     notes / para .bool_set:N = \l_@@_notes_para_bool ,
855     notes / para .default:n = true ,
856     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
857     notes / bottomrule .default:n = true ,
858     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
859     tabularnote .value_required:n = true ,
860     last-col .code:n = \tl_if_empty:nF { #1 }

```

```

861             { \@@_error:n { last-col-non-empty-for-NiceArray } }
862             \int_zero:N \l_@@_last_col_int ,
863     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
864     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
865     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
866 }

```

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}`).

```

867 \cs_new_protected:Npn \@@_cell_begin:w
868 {

```

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*).

```

869     \tl_gclear:N \g_@@_post_action_cell_tl

```

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

```

870     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

871     \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.

```

872     \int_compare:nNnT \c@jCol = 1
873     { \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).

```

874     \hbox_set:Nw \l_@@_cell_box
875     \bool_if:NF \l_@@_NiceTabular_bool
876     {
877         \c_math_toggle_token
878         \bool_if:NT \l_@@_small_bool \scriptstyle
879     }

```

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).

```

880     \color { nicematrix }
881     \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.

```

882     \int_compare:nNnTF \c@iRow = 0
883     {
884         \int_compare:nNnT \c@jCol > 0
885         {
886             \l_@@_code_for_first_row_tl
887             \xglobal \colorlet { nicematrix-first-row } { . }
888         }
889     }
890     {

```

```

891     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
892     {
893         \l_@@_code_for_last_row_tl
894         \xglobal \colorlet { nicematrix-last-row } { . }
895     }
896 }
897 }

```

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.

```

898 \cs_new_protected:Npn \@@_begin_of_row:
899 {
900     \int_gincr:N \c@iRow
901     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
902     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
903     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
904     \pgfpicture
905     \pgfrememberpicturepositiononpagetrue
906     \pgfcoordinate
907     { \@@_env: - row - \int_use:N \c@iRow - base }
908     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
909     \str_if_empty:NF \l_@@_name_str
910     {
911         \pgfnodealias
912         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
913         { \@@_env: - row - \int_use:N \c@iRow - base }
914     }
915     \endpgfpicture
916 }

```

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.

```

917 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
918 {
919     \int_compare:nNnTF \c@iRow = 0
920     {
921         \dim_gset:Nn \g_@@_dp_row_zero_dim
922         { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
923         \dim_gset:Nn \g_@@_ht_row_zero_dim
924         { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
925     }
926     {
927         \int_compare:nNnT \c@iRow = 1
928         {
929             \dim_gset:Nn \g_@@_ht_row_one_dim
930             { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
931         }
932     }
933 }
934 \cs_new_protected:Npn \@@_rotate_cell_box:
935 {
936     \box_rotate:Nn \l_@@_cell_box { 90 }
937     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
938     {
939         \vbox_set_top:Nn \l_@@_cell_box
940         {
941             \vbox_to_zero:n { }

```

```

942         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
943         \box_use:N \l_@@_cell_box
944     }
945 }
946 \bool_gset_false:N \g_@@_rotate_bool
947 }
948 \cs_new_protected:Npn \@@_adjust_size_box:
949 {
950     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
951     {
952         \box_set_wd:Nn \l_@@_cell_box
953         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
954         \dim_gzero:N \g_@@_blocks_wd_dim
955     }
956     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
957     {
958         \box_set_dp:Nn \l_@@_cell_box
959         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
960         \dim_gzero:N \g_@@_blocks_dp_dim
961     }
962     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
963     {
964         \box_set_ht:Nn \l_@@_cell_box
965         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
966         \dim_gzero:N \g_@@_blocks_ht_dim
967     }
968 }
969 \cs_new_protected:Npn \@@_cell_end:
970 {
971     \@@_math_toggle_token:
972     \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.

```

973     \g_@@_post_action_cell_tl
974     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
975     \@@_adjust_size_box:
976     \box_set_ht:Nn \l_@@_cell_box
977     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
978     \box_set_dp:Nn \l_@@_cell_box
979     { \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”).

```

980     \dim_gset:Nn \g_@@_max_cell_width_dim
981     { \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”.

```

982     \@@_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.

```

983 \bool_if:NTF \g_@@_empty_cell_bool
984 { \box_use_drop:N \l_@@_cell_box }
985 {
986   \bool_lazy_or:nnTF
987     \g_@@_not_empty_cell_bool
988     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
989     \@@_node_for_cell:
990     { \box_use_drop:N \l_@@_cell_box }
991 }
992 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
993 \bool_gset_false:N \g_@@_empty_cell_bool
994 \bool_gset_false:N \g_@@_not_empty_cell_bool
995 }

```

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

```

996 \cs_new_protected:Npn \@@_node_for_cell:
997 {
998   \pgfpicture
999   \pgfsetbaseline \c_zero_dim
1000   \pgfrememberpicturepositiononpagetrue
1001   \pgfset
1002   {
1003     inner~sep = \c_zero_dim ,
1004     minimum~width = \c_zero_dim
1005   }
1006   \pgfnode
1007   { rectangle }
1008   { base }
1009   { \box_use_drop:N \l_@@_cell_box }
1010   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1011   { }
1012   \str_if_empty:NF \l_@@_name_str
1013   {
1014     \pgfnodealias
1015     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1016     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1017   }
1018   \endpgfpicture
1019 }

```

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.

```

1020 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1021 {
1022   \cs_new_protected:Npn \@@_patch_node_for_cell:
1023   {
1024     \hbox_set:Nn \l_@@_cell_box
1025     {
1026       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1027       \hbox_overlap_left:n
1028       {
1029         \pgfsys@markposition
1030         { \@@_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.

```

1031         #1
1032     }
1033     \box_use:N \l_@@_cell_box
1034     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1035     \hbox_overlap_left:n
1036     {
1037         \pgfsys@markposition
1038         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1039         #1
1040     }
1041 }
1042 }
1043 }

```

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

```

1044 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1045 {
1046     \@@_patch_node_for_cell:n
1047     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1048 }
1049 { \@@_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.

```

1050 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1051 {
1052     \bool_if:nTF { #1 } { \tl_gput_left:cx \tl_gput_right:cx
1053         { \g_@@_#2 _ lines _ tl }
1054         {
1055             \use:c { @@ _ draw _ #2 : nnn }
1056             { \int_use:N \c@iRow }
1057             { \int_use:N \c@jCol }
1058             { \exp_not:n { #3 } }
1059         }
1060     }

1061 \cs_new_protected:Npn \@@_array:
1062 {
1063     \bool_if:NTF \l_@@_NiceTabular_bool
1064     { \dim_set_eq:NN \col@sep \tabcolsep }
1065     { \dim_set_eq:NN \col@sep \arraycolsep }
1066     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1067     { \cs_set_nopar:Npn \@halignto { } }
1068     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1069 \@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.

```
1070   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1071 }
```

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.

```
1072 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1073 \cs_new_protected:Npn \@@_create_row_node:
1074 {
```

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

```
1075   \hbox
1076   {
1077     \bool_if:NT \l_@@_code_before_bool
1078     {
1079       \vtop
1080       {
1081         \skip_vertical:N 0.5\arrayrulewidth
1082         \pgfsys@markposition
1083         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1084         \skip_vertical:N -0.5\arrayrulewidth
1085       }
1086     }
1087     \pgfpicture
1088     \pgfrememberpicturepositiononpagetrue
1089     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1090     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1091     \str_if_empty:NF \l_@@_name_str
1092     {
1093       \pgfnodealias
1094       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1095       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1096     }
1097     \endpgfpicture
1098   }
1099 }
```

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

```
1100 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1101 \cs_new_protected:Npn \@@_everycr_i:
1102 {
1103   \int_gzero:N \c@jCol
1104   \bool_gset_false:N \g_@@_after_col_zero_bool
1105   \bool_if:NF \g_@@_row_of_col_done_bool
1106   {
1107     \@@_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).

```
1108   \tl_if_empty:NF \l_@@_hlines_clist
1109   {
1110     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1111     {
1112       \exp_args:NNx
1113       \clist_if_in:NnT
1114       \l_@@_hlines_clist
1115       { \int_eval:n { \c@iRow + 1 } }
1116     }
1117   }
```


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.

```

1118         \int_compare:nNnT \c@iRow > { -1 }
1119         {
1120             \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@`.

```

1121             { \hrule height \arrayrulewidth width \c_zero_dim }
1122         }
1123     }
1124 }
1125 }
1126 }

```

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`).

```

1127 \cs_set_protected:Npn \@@_newcolumntype #1
1128 {
1129     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1130     \peek_meaning:NTF [
1131         { \newcol@ #1 }
1132         { \newcol@ #1 [ 0 ] }
1133     }

```

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

```

1134 \cs_set_protected:Npn \@@_renew_dots:
1135 {
1136     \cs_set_eq:NN \ldots \@@_Ldots
1137     \cs_set_eq:NN \cdots \@@_Cdots
1138     \cs_set_eq:NN \vdots \@@_Vdots
1139     \cs_set_eq:NN \ddots \@@_Ddots
1140     \cs_set_eq:NN \iddots \@@_Iddots
1141     \cs_set_eq:NN \dots \@@_Ldots
1142     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1143 }

```

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

```

1144 \cs_new_protected:Npn \@@_colortbl_like:
1145 {
1146     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1147     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1148     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1149 }

```

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

```

1150 \cs_new_protected:Npn \@@_pre_array_ii:
1151 {

```

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.

```

1152     \xglobal \colorlet { nicematrix } { . }

```

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

```

1153     \int_gzero:N \g_@@_total_X_weight_int
1154     \@@_expand_clist:N \l_@@_hlines_clist
1155     \@@_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.

```

1156 \bool_if:NT \c_@@_booktabs_loaded_bool
1157 { \tl_put_left:Nn \@BTnormal \@_create_row_node: }
1158 \box_clear_new:N \l_@@_cell_box
1159 \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}`).

```

1160 \bool_if:NT \l_@@_small_bool
1161 {
1162     \cs_set_nopar:Npn \arraystretch { 0.47 }
1163     \dim_set:Nn \arraycolsep { 1.45 pt }
1164 }

1165 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1166 {
1167     \tl_put_right:Nn \@_begin_of_row:
1168     {
1169         \pgfsys@markposition
1170         { \@_env: - row - \int_use:N \c@iRow - base }
1171     }
1172 }

```

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`.

```

1173 \cs_set_nopar:Npn \ialign
1174 {
1175     \bool_if:NTF \c_@@_colortbl_loaded_bool
1176     {
1177         \CT@everycr
1178         {
1179             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1180             \@_everycr:
1181         }
1182     }
1183     { \everycr { \@_everycr: } }
1184     \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`.

```

1185 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1186 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1187 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1188 \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}`.

```

1189 \dim_gzero_new:N \g_@@_ht_row_one_dim
1190 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1191 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1192 \dim_gzero_new:N \g_@@_ht_last_row_dim
1193 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1194 \dim_gzero_new:N \g_@@_dp_last_row_dim
1195 \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`.

```

1196 \cs_set_eq:NN \ialign \@@_old_ialign:
1197 \halign
1198 }

```

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).

```

1199 \cs_set_eq:NN \@@_old_ldots \ldots
1200 \cs_set_eq:NN \@@_old_cdots \cdots
1201 \cs_set_eq:NN \@@_old_vdots \vdots
1202 \cs_set_eq:NN \@@_old_ddots \ddots
1203 \cs_set_eq:NN \@@_old_iddots \iddots
1204 \bool_if:NTF \l_@@_standard_cline_bool
1205 { \cs_set_eq:NN \cline \@@_standard_cline }
1206 { \cs_set_eq:NN \cline \@@_cline }
1207 \cs_set_eq:NN \Ldots \@@_Ldots
1208 \cs_set_eq:NN \Cdots \@@_Cdots
1209 \cs_set_eq:NN \Vdots \@@_Vdots
1210 \cs_set_eq:NN \Ddots \@@_Ddots
1211 \cs_set_eq:NN \Iddots \@@_Iddots
1212 \cs_set_eq:NN \hdottedline \@@_hdottedline:
1213 \cs_set_eq:NN \Hline \@@_Hline:
1214 \cs_set_eq:NN \Hspace \@@_Hspace:
1215 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1216 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1217 \cs_set_eq:NN \Block \@@_Block:
1218 \cs_set_eq:NN \rotate \@@_rotate:
1219 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1220 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1221 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1222 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1223 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1224 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1225 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1226 \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.

```

1227 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1228 \hook_gput_code:nnn { env / tabular / begin } { . }
1229 { \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).

```

1230 \seq_gclear:N \g_@@_multicolumn_cells_seq
1231 \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).

```

1232 \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`).

```
1233 \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.

```
1234 \int_gzero_new:N \g_@@_col_total_int
```

```
1235 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

```
1236 \@@_renew_NC@rewrite@S:
```

```
1237 \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.

```
1238 \tl_gclear_new:N \g_@@_Cdots_lines_tl
```

```
1239 \tl_gclear_new:N \g_@@_Ldots_lines_tl
```

```
1240 \tl_gclear_new:N \g_@@_Vdots_lines_tl
```

```
1241 \tl_gclear_new:N \g_@@_Ddots_lines_tl
```

```
1242 \tl_gclear_new:N \g_@@_Iddots_lines_tl
```

```
1243 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl
```

```
1244 \tl_gclear_new:N \g_nicematrix_code_before_tl
```

```
1245 }
```

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

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

```
1246 \cs_new_protected:Npn \@@_pre_array:
```

```
1247 {
```

```
1248 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
```

```
1249 \int_gzero_new:N \c@iRow
```

```
1250 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
```

```
1251 \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).

```
1252 \int_compare:nNnT \l_@@_last_row_int = { -1 }
```

```
1253 {
```

```
1254 \bool_set_true:N \l_@@_last_row_without_value_bool
```

```
1255 \bool_if:NT \g_@@_aux_found_bool
```

```
1256 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \c_@@_size_seq 3 } }
```

```
1257 }
```

```
1258 \int_compare:nNnT \l_@@_last_col_int = { -1 }
```

```
1259 {
```

```
1260 \bool_if:NT \g_@@_aux_found_bool
```

```
1261 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \c_@@_size_seq 6 } }
```

```
1262 }
```

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”.

```
1263 \int_compare:nNnT \l_@@_last_row_int > { -2 }
```

```
1264 {
```

```
1265 \tl_put_right:Nn \@@_update_for_first_and_last_row:
```

```
1266 {
```

```

1267         \dim_gset:Nn \g_@@_ht_last_row_dim
1268         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1269         \dim_gset:Nn \g_@@_dp_last_row_dim
1270         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1271     }
1272 }

1273 \seq_gclear:N \g_@@_cols_vlism_seq
1274 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1275 \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.

```

1276 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1277 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1278 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1279 \@@_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.

```

1280 \box_clear_new:N \l_@@_the_array_box

```

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

```

1281 \@@_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]`).

```

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

```

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

```

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

```

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).

```

1295 \hbox_set:Nw \l_@@_the_array_box

```

```

1296 \skip_horizontal:N \l_@@_left_margin_dim
1297 \skip_horizontal:N \l_@@_extra_left_margin_dim
1298 \c_math_toggle_token
1299 \bool_if:NTF \l_@@_light_syntax_bool
1300 { \use:c { @@-light-syntax } }
1301 { \use:c { @@-normal-syntax } }
1302 }

```

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

```

1303 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1304 {
1305   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1306   \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...

```

1307 \@@_pre_array:
1308 }

```

The `\CodeBefore`

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

```

1309 \cs_new_protected:Npn \@@_pre_code_before:
1310 {

```

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).

```

1311 \int_set:Nn \c@iRow { \seq_item:Nn \c_@@_size_seq 2 }
1312 \int_set:Nn \c@jCol { \seq_item:Nn \c_@@_size_seq 5 }
1313 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \c_@@_size_seq 3 }
1314 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \c_@@_size_seq 6 }

```

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

```

1315 \pgfsys@markposition { \@@_env: - position }
1316 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1317 \pgfpicture
1318 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1319 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1320 {
1321   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1322   \pgfcoordinate { \@@_env: - row - ##1 }
1323   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1324 }

```

Now, the recreation of the `col` nodes.

```

1325 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1326 {
1327   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1328   \pgfcoordinate { \@@_env: - col - ##1 }
1329   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1330 }

```

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

```

1331 \@@_create_diag_nodes:

```

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

```
1332 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1333 \endpgfpicture
```

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

```
1334 \@@_create_blocks_nodes:
1335 \bool_if:NT \c_@@_tikz_loaded_bool
1336 {
1337   \tikzset
1338   {
1339     every~picture / .style =
1340     { overlay , name~prefix = \@@_env: - }
1341   }
1342 }
1343 \cs_set_eq:NN \cellcolor \@@_cellcolor
1344 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1345 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1346 \cs_set_eq:NN \rowcolor \@@_rowcolor
1347 \cs_set_eq:NN \rowcolors \@@_rowcolors
1348 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1349 \cs_set_eq:NN \arraycolor \@@_arraycolor
1350 \cs_set_eq:NN \columncolor \@@_columncolor
1351 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1352 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1353 }
```

```
1354 \cs_new_protected:Npn \@@_exec_code_before:
1355 {
1356   \seq_gclear_new:N \g_@@_colors_seq
1357   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1358   \group_begin:
```

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

```
1359 \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`.

```
1360 \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.

```
1361 \@@_actually_color:
1362 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1363 \group_end:
1364 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1365 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1366 }
```

```
1367 \keys_define:nn { NiceMatrix / CodeBefore }
1368 {
1369   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1370   create-cell-nodes .default:n = true ,
1371   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1372   sub-matrix .value_required:n = true ,
1373   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
```

```

1374     delimiters / color .value_required:n = true ,
1375     unknown .code:n = \@@_error:n { Unknown-key-for~CodeAfter }
1376 }

1377 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1378 {
1379     \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1380     \@@_CodeBefore:w
1381 }

```

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.

```

1382 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1383 {
1384     \bool_if:NT \g_@@_aux_found_bool
1385     {
1386         \@@_pre_code_before:
1387         #1
1388     }
1389 }

```

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.

```

1390 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1391 {
1392     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1393     {
1394         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1395         \pgfcoordinate { \@@_env: - row - ##1 - base }
1396         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1397         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1398         {
1399             \cs_if_exist:cT
1400             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1401             {
1402                 \pgfsys@getposition
1403                 { \@@_env: - ##1 - ####1 - NW }
1404                 \@@_node_position:
1405                 \pgfsys@getposition
1406                 { \@@_env: - ##1 - ####1 - SE }
1407                 \@@_node_position_i:
1408                 \@@_pgf_rect_node:nnn
1409                 { \@@_env: - ##1 - ####1 }
1410                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1411                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1412             }
1413         }
1414     }
1415     \int_step_inline:nn \c@iRow
1416     {
1417         \pgfnodealias
1418         { \@@_env: - ##1 - last }
1419         { \@@_env: - ##1 - \int_use:N \c@jCol }
1420     }
1421     \int_step_inline:nn \c@jCol
1422     {
1423         \pgfnodealias
1424         { \@@_env: - last - ##1 }
1425         { \@@_env: - \int_use:N \c@iRow - ##1 }

```



```

1426     }
1427     \@@_create_extra_nodes:
1428 }

1429 \cs_new_protected:Npn \@@_create_blocks_nodes:
1430 {
1431     \pgfpicture
1432     \pgf@relevantforpicturesizefalse
1433     \pgfrememberpicturepositiononpagetrue
1434     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1435     { \@@_create_one_block_node:nnnnn #1 }
1436     \endpgfpicture
1437 }

```

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

```

1438 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1439 {
1440     \tl_if_empty:nF { #5 }
1441     {
1442         \@@_qpoint:n { col - #2 }
1443         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1444         \@@_qpoint:n { #1 }
1445         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1446         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1447         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1448         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1449         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1450         \@@_pgf_rect_node:nnnnn
1451         { \@@_env: - #5 }
1452         { \dim_use:N \l_tmpa_dim }
1453         { \dim_use:N \l_tmpb_dim }
1454         { \dim_use:N \l_@@_tmpc_dim }
1455         { \dim_use:N \l_@@_tmpd_dim }
1456     }
1457 }

1458 \cs_new_protected:Npn \@@_patch_for_revtext:
1459 {
1460     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1461     \cs_set_eq:NN \insert@column \insert@column@array
1462     \cs_set_eq:NN \@classx \@classx@array
1463     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1464     \cs_set_eq:NN \@arraycr \@arraycr@array
1465     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1466     \cs_set_eq:NN \array \array@array
1467     \cs_set_eq:NN \@array \@array@array
1468     \cs_set_eq:NN \@tabular \@tabular@array
1469     \cs_set_eq:NN \@mkpream \@mkpream@array
1470     \cs_set_eq:NN \endarray \endarray@array
1471     \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1472     \cs_set:Npn \endtabular { \endarray $\egroup} % $
1473 }

```

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).

The environment {NiceArrayWithDelims}

```

1474 \NewDocumentEnvironment { NiceArrayWithDelims }
1475 { m m O { } m ! O { } t \CodeBefore }
1476 {
1477   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1478   \@@_provide_pgfsyspdfmark:
1479   \bool_if:NT \c_@@_footnote_bool \savenotes

The aim of the following \bgroup (the corresponding \egroup is, of course, at the end of the envi-
ronment) is to be able to put an exposant to a matrix in a mathematical formula.
1480   \bgroup

1481   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1482   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1483   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1484   \int_gzero:N \g_@@_block_box_int
1485   \dim_zero:N \g_@@_width_last_col_dim
1486   \dim_zero:N \g_@@_width_first_col_dim
1487   \bool_gset_false:N \g_@@_row_of_col_done_bool
1488   \str_if_empty:NT \g_@@_name_env_str
1489     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1490   \bool_if:NTF \l_@@_NiceTabular_bool
1491     \mode_leave_vertical:
1492     \@@_test_if_math_mode:
1493   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1494   \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.

```

1495   \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.
1496   \cs_if_exist:NT \tikz@library@external@loaded
1497   {
1498     \tikzexternaldisable
1499     \cs_if_exist:NT \ifstandalone
1500       { \tikzset { external / optimize = false } }
1501   }

```

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

```

1502   \int_gincr:N \g_@@_env_int
1503   \bool_if:NF \l_@@_block_auto_columns_width_bool
1504     { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

The sequence `\g_@@_blocks_seq` will contain the carateristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks (except the blocks with the key `hvlines`).

```

1505   \seq_gclear:N \g_@@_blocks_seq
1506   \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`.

```

1507   \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1508   \seq_gclear:N \g_@@_pos_of_xdots_seq
1509   \tl_gclear_new:N \g_@@_code_before_tl

```

⁶⁵e.g. `\color[rgb]{0.5,0.5,0}`

```
1510 \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.

```
1511 \bool_gset_false:N \g_@@_aux_found_bool
1512 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1513 {
1514   \bool_gset_true:N \g_@@_aux_found_bool
1515   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1516 }
```

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.

```
1517 \tl_gclear:N \g_@@_aux_tl
1518 \tl_if_empty:NF \g_@@_code_before_tl
1519 {
1520   \bool_set_true:N \l_@@_code_before_bool
1521   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1522 }
```

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.

```
1523 \bool_if:NTF \l_@@_NiceArray_bool
1524 { \keys_set:nn { NiceMatrix / NiceArray } }
1525 { \keys_set:nn { NiceMatrix / pNiceArray } }
1526 { #3 , #5 }

1527 \tl_if_empty:NF \l_@@_rules_color_tl
1528 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
```

The argument #6 is the last argument of {NiceArrayWithDelims}. With that argument of type “t \CodeBefore”, we test whether there is the keyword \CodeBefore at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword \CodeBefore and the (other) keyword \Body. It’s the job that will do the command \@@_pre_array_i:w. After that job, the command \@@_pre_array_i:w will go on with \@@_pre_array:.

```
1529 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
1530 }
1531 {
1532   \bool_if:NTF \l_@@_light_syntax_bool
1533   { \use:c { end @@-light-syntax } }
1534   { \use:c { end @@-normal-syntax } }
1535   \c_math_toggle_token
1536   \skip_horizontal:N \l_@@_right_margin_dim
1537   \skip_horizontal:N \l_@@_extra_right_margin_dim
1538   \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.

```
1539 \bool_if:NT \l_@@_width_used_bool
1540 {
1541   \int_compare:nNt \g_@@_total_X_weight_int = 0
1542   { \@@_error:n { width~without~X~columns } }
1543 }
```

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 .

```
1544 \int_compare:nNt \g_@@_total_X_weight_int > 0
1545 {
```

```

1546 \tl_gput_right:Nx \g_@@_aux_tl
1547 {
1548   \bool_set_true:N \l_@@_X_columns_aux_bool
1549   \dim_set:Nn \l_@@_X_columns_dim
1550   {
1551     \dim_compare:nNnTF
1552     {
1553       \dim_abs:n
1554       { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1555     }
1556     <
1557     { 0.001 pt }
1558     { \dim_use:N \l_@@_X_columns_dim }
1559     {
1560       \dim_eval:n
1561       {
1562         ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1563         / \int_use:N \g_@@_total_X_weight_int
1564         + \l_@@_X_columns_dim
1565       }
1566     }
1567   }
1568 }
1569 }

```

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

```

1570 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1571 {
1572   \bool_if:NF \l_@@_last_row_without_value_bool
1573   {
1574     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1575     {
1576       \@@_error:n { Wrong~last~row }
1577       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1578     }
1579   }
1580 }

```

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

```

1581 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1582 \bool_if:nTF \g_@@_last_col_found_bool
1583 { \int_gdecr:N \c@jCol }
1584 {
1585   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1586   { \@@_error:n { last~col~not~used } }
1587 }

```

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

```

1588 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1589 \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. 128).

```

1590 \int_compare:nNnT \l_@@_first_col_int = 0
1591 {
1592   \skip_horizontal:N \col@sep
1593   \skip_horizontal:N \g_@@_width_first_col_dim

```

⁶⁶We remind that the potential “first column” (exterior) has the number 0.

```
1594 }
```

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.

```
1595 \bool_if:NTF \l_@@_NiceArray_bool
1596 {
1597   \str_case:VnF \l_@@_baseline_tl
1598   {
1599     b \@@_use_arraybox_with_notes_b:
1600     c \@@_use_arraybox_with_notes_c:
1601   }
1602   \@@_use_arraybox_with_notes:
1603 }
```

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).

```
1604 {
1605   \int_compare:nNnTF \l_@@_first_row_int = 0
1606   {
1607     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1608     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1609   }
1610   { \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

```
1611 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1612 {
1613   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1614   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1615 }
1616 { \dim_zero:N \l_tmpb_dim }
1617 \hbox_set:Nn \l_tmpa_box
1618 {
1619   \c_math_toggle_token
1620   \tl_if_empty:NF \l_@@_delimiters_color_tl
1621   { \color { \l_@@_delimiters_color_tl } }
1622   \exp_after:wN \left \g_@@_left_delim_tl
1623   \vcenter
1624   {
```

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).

```
1625   \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1626   \hbox
1627   {
1628     \bool_if:NTF \l_@@_NiceTabular_bool
1629     { \skip_horizontal:N -\tabcolsep }
1630     { \skip_horizontal:N -\arraycolsep }
1631     \@@_use_arraybox_with_notes_c:
1632     \bool_if:NTF \l_@@_NiceTabular_bool
1633     { \skip_horizontal:N -\tabcolsep }
1634     { \skip_horizontal:N -\arraycolsep }
1635   }
```

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).

```
1636   \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1637 }
```

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).

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.

```

1638         \tl_if_empty:NF \l_@@_delimiters_color_tl
1639         { \color { \l_@@_delimiters_color_tl } }
1640         \exp_after:wN \right \g_@@_right_delim_tl
1641         \c_math_toggle_token
1642     }

```

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.

```

1643     \bool_if:NTF \l_@@_delimiters_max_width_bool
1644     {
1645         \@@_put_box_in_flow_bis:nn
1646         \g_@@_left_delim_tl \g_@@_right_delim_tl
1647     }
1648     \@@_put_box_in_flow:
1649 }

```

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. 129).

```

1650     \bool_if:NT \g_@@_last_col_found_bool
1651     {
1652         \skip_horizontal:N \g_@@_width_last_col_dim
1653         \skip_horizontal:N \col@sep
1654     }
1655     \bool_if:NF \l_@@_Matrix_bool
1656     {
1657         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1658         { \@@_error:n { columns-not-used } }
1659     }
1660     \group_begin:
1661     \globaldefs = 1
1662     \@@_msg_redirect_name:nn { columns-not-used } { error }
1663     \group_end:
1664     \@@_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.

```

1665     \egroup

```

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

```

1666     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1667     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1668     \iow_now:Nx \@mainaux
1669     {
1670         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
1671         { \exp_not:V \g_@@_aux_tl }
1672     }
1673     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1674     \bool_if:NT \c_@@_footnote_bool \endsavenotes
1675 }

```

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.

```

1676 \cs_new_protected:Npn \@@_construct_preamble:
1677 {

```

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.

```

1678 \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).

```

1679 \bool_if:NF \l_@@_Matrix_bool
1680 {
1681   \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1682   \@@_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).

```

1683 \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).

```

1684 \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1685 \@tempswatrue

```

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

```

1686 \@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`.

```

1687 \int_gzero:N \c@jCol
1688 \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1689 \bool_gset_false:N \g_tmpb_bool
1690 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1691 {
1692   \tl_gset:Nn \g_@@_preamble_tl
1693   { ! { \skip_horizontal:N \arrayrulewidth } }
1694 }
1695 {
1696   \clist_if_in:NnT \l_@@_vlines_clist 1
1697   {
1698     \tl_gset:Nn \g_@@_preamble_tl
1699     { ! { \skip_horizontal:N \arrayrulewidth } }
1700   }
1701 }

```

The sequence `\g_@@_cols_vlism_seq` will contain the numbers of the columns where you will to have to draw vertical lines in the potential sub-matrices (hence the name `vlism`).

```

1702 \seq_clear:N \g_@@_cols_vlism_seq

```

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

```
1703 \int_zero:N \l_tmpa_int
```

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

```
1704 \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1705 \int_gset_eq:NN \g_@@_static_num_of_col_int \c_jCol
1706 }
```

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

```
1707 \bool_if:NT \l_@@_colortbl_like_bool
1708 {
1709   \regex_replace_all:NnN
1710   \c_@@_columncolor_regex
1711   { \c { @@_columncolor_preamble } }
1712   \g_@@_preamble_tl
1713 }
```

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

```
1714 \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}`.

```
1715 \bool_lazy_or:nnT
1716 { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1717 { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1718 { \bool_set_false:N \l_@@_NiceArray_bool }
```

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

```
1719 \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).

```
1720 \int_compare:nNnTF \l_@@_first_col_int = 0
1721 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1722 {
1723   \bool_lazy_all:nT
1724   {
1725     \l_@@_NiceArray_bool
1726     { \bool_not_p:n \l_@@_NiceTabular_bool }
1727     { \tl_if_empty_p:N \l_@@_vlines_clist }
1728     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1729   }
1730   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1731 }
1732 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1733 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1734 {
1735   \bool_lazy_all:nT
1736   {
1737     \l_@@_NiceArray_bool
1738     { \bool_not_p:n \l_@@_NiceTabular_bool }
1739     { \tl_if_empty_p:N \l_@@_vlines_clist }
1740     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1741   }
1742   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1743 }
```

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`).

```
1744 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1745 {
1746   \tl_gput_right:Nn \g_@@_preamble_tl
```



```

1747         { > { \@@_error_too_much_cols: } 1 }
1748     }
1749 }

```

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

```

1750 \cs_new_protected:Npn \@@_patch_preamble:n #1
1751 {
1752     \str_case:nnF { #1 }
1753     {
1754         c { \@@_patch_preamble_i:n #1 }
1755         l { \@@_patch_preamble_i:n #1 }
1756         r { \@@_patch_preamble_i:n #1 }
1757         > { \@@_patch_preamble_ii:nn #1 }
1758         ! { \@@_patch_preamble_ii:nn #1 }
1759         @ { \@@_patch_preamble_ii:nn #1 }
1760         | { \@@_patch_preamble_iii:n #1 }
1761         p { \@@_patch_preamble_iv:n #1 }
1762         b { \@@_patch_preamble_iv:n #1 }
1763         m { \@@_patch_preamble_iv:n #1 }
1764         \@@_V: { \@@_patch_preamble_v:n }
1765         V { \@@_patch_preamble_v:n }
1766         \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
1767         \@@_W: { \@@_patch_preamble_vi:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1768         \@@_S: { \@@_patch_preamble_vii:n }
1769         ( { \@@_patch_preamble_viii:nn #1 }
1770         [ { \@@_patch_preamble_viii:nn #1 }
1771         \{ { \@@_patch_preamble_viii:nn #1 }
1772         ) { \@@_patch_preamble_ix:nn #1 }
1773         ] { \@@_patch_preamble_ix:nn #1 }
1774         \} { \@@_patch_preamble_ix:nn #1 }
1775         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}`.

```

1776     \@@_X { \@@_patch_preamble_x:n }
1777     \q_stop { }
1778 }
1779 {
1780     \str_case:e:nnF { #1 }
1781     {
1782         \l_@@_letter_for_dotted_lines_str { \@@_patch_preamble_xii:n #1 }
1783         \l_@@_letter_vlism_tl
1784         {
1785             \seq_gput_right:Nx \g_@@_cols_vlism_seq
1786             { \int_eval:n { \c@jCol + 1 } }
1787             \tl_gput_right:Nx \g_@@_preamble_tl
1788             { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1789             \@@_patch_preamble:n
1790         }
1791         { : }
1792         {
1793             \bool_if:NTF \c_@@_arydshln_loaded_bool
1794             {
1795                 \tl_gput_right:Nn \g_@@_preamble_tl { : }
1796                 \@@_patch_preamble:n
1797             }
1798             { \@@_fatal:n { colon~without~arydshln } }
1799         }
1800     }

```

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).

```

1801      {
1802      \keys_set_known:nnN { NiceMatrix / ColumnTypes } { #1 } \l_tmpa_tl
1803      \tl_if_empty:NTF \l_tmpa_tl
1804      \@@_patch_preamble:n
1805      { \@@_fatal:nn { unknown~column~type } { #1 } }
1806      }
1807    }
1808  }

```

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

```

1809 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1810 {
1811   \tl_gput_right:Nn \g_@@_preamble_tl
1812   {
1813     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
1814     #1
1815     < \@@_cell_end:
1816   }

```

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

```

1817   \int_gincr:N \c@jCol
1818   \@@_patch_preamble_xi:n
1819 }

```

For >, ! and @

```

1820 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1821 {
1822   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1823   \@@_patch_preamble:n
1824 }

```

For |

```

1825 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1826 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1827   \int_incr:N \l_tmpa_int
1828   \@@_patch_preamble_iii_i:n
1829 }
1830 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1831 {
1832   \str_if_eq:nnTF { #1 } |
1833   { \@@_patch_preamble_iii:n | }
1834   {
1835     \tl_gput_right:Nx \g_@@_preamble_tl
1836     {
1837       \exp_not:N !
1838       {
1839         \skip_horizontal:n
1840         {

```

Here, the command \dim_eval:n is mandatory.

```

1841         \dim_eval:n
1842         {
1843           \arrayrulewidth * \l_tmpa_int
1844           + \doublerulesep * ( \l_tmpa_int - 1)

```

```

1845     }
1846   }
1847 }
1848 }
1849 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1850 {
1851   \@@_vline:n
1852   {
1853     position = \int_eval:n { \c@jCol + 1 } ,
1854     multiplicity = \int_use:N \l_tmpa_int ,
1855   }

```

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.

```

1856   }
1857   \int_zero:N \l_tmpa_int
1858   \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
1859   \@@_patch_preamble:n #1
1860 }
1861 }
1862 \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.

```

1863 \keys_define:nn { WithArrows / p-column }
1864 {
1865   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
1866   r .value_forbidden:n = true ,
1867   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
1868   c .value_forbidden:n = true ,
1869   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
1870   l .value_forbidden:n = true ,
1871   si .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
1872   si .value_forbidden:n = true ,
1873   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
1874   p .value_forbidden:n = true ,
1875   t .meta:n = p ,
1876   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
1877   m .value_forbidden:n = true ,
1878   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
1879   b .value_forbidden:n = true ,
1880 }

```

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

```

1881 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
1882 {
1883   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

1884   \@@_patch_preamble_iv_i:n
1885 }
1886 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
1887 {
1888   \str_if_eq:nnTF { #1 } { [ }
1889   { \@@_patch_preamble_iv_ii:w [ }
1890   { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
1891 }
1892 \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
1893 { \@@_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.

```
1894 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
1895 {
```

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).

```
1896   \str_set:Nn \l_@@_hpos_col_str { j }
1897   \keys_set:nn { WithArrows / p-column } { #1 }
1898   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
1899 }
```

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

```
1900 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
1901 {
1902   \use:x
1903   {
1904     \@@_patch_preamble_iv_v:nnnnnnnn
1905     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
1906     { \dim_eval:n { #1 } }
1907   }
```

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.

```
1908       \str_if_eq:VnTF \l_@@_hpos_col_str j
1909       { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
1910       {
1911         \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
1912         { \l_@@_hpos_col_str }
1913       }
1914       \str_case:Vn \l_@@_hpos_col_str
1915       {
1916         c { \exp_not:N \centering }
1917         l { \exp_not:N \raggedright }
1918         r { \exp_not:N \raggedleft }
1919       }
1920     }
1921     { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
1922     { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
1923     { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
1924     { #2 }
1925     {
1926       \str_case:VnF \l_@@_hpos_col_str
1927       {
1928         { j } { c }
1929         { si } { c }
1930       }
1931       { \l_@@_hpos_col_str }
1932     }
1933   }
```

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

```
1934   \int_gincr:N \c@jCol
1935   \@@_patch_preamble_xi:n
1936 }
```

#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*.

```

1937 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
1938 {
1939   \tl_gput_right:Nn \g_@@_preamble_tl
1940   {
1941     > {

```

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.

```

1942       \dim_set:Nn \l_@@_col_width_dim { #2 }
1943       \@@_cell_begin:w
1944       \begin { #7 } [ #1 ] { #2 }

```

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

```

1945       \everypar
1946       {
1947         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
1948         \everypar { }
1949       }

```

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

```

1950       #3

```

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

```

1951       \g_@@_row_style_tl
1952       \arraybackslash
1953       #5
1954     }
1955     #8
1956     < {
1957       #6

```

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

```

1958       \@finalstrut \@arstrutbox
1959       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
1960       \end { #7 }

```

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

```

1961       #4
1962       \@@_cell_end:
1963     }
1964   }
1965 }

```

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.

```

1966 \cs_new_protected:Npn \@@_center_cell_box:
1967 {

```

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

```

1968 \tl_gput_right:Nn \g_@@_post_action_cell_tl
1969 {
1970   \int_compare:nNnT
1971     { \box_ht:N \l_@@_cell_box }
1972     >
1973     { \box_ht:N \@arstrutbox }
1974     {
1975       \hbox_set:Nn \l_@@_cell_box
1976       {
1977         \box_move_down:nn
1978         {
1979           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
1980             + \baselineskip ) / 2
1981         }
1982         { \box_use:N \l_@@_cell_box }
1983       }
1984     }
1985   }
1986 }

```

For `V` (similar to the `V` of `varwidth`).

```

1987 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
1988 {
1989   \str_if_eq:nnTF { #1 } { [ ] }
1990   { \@@_patch_preamble_v_i:w [ ] }
1991   { \@@_patch_preamble_v_i:w [ ] { #1 } }
1992 }
1993 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
1994 { \@@_patch_preamble_v_ii:nn { #1 } }
1995 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
1996 {
1997   \str_set:Nn \l_@@_vpos_col_str { p }
1998   \str_set:Nn \l_@@_hpos_col_str { j }
1999   \keys_set:nn { WithArrows / p-column } { #1 }
2000   \bool_if:NTF \c_@@_varwidth_loaded_bool
2001   { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2002   {
2003     \@@_error:n { varwidth~not~loaded }
2004     \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2005   }
2006 }

```

For `w` and `W`

```

2007 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2008 {
2009   \tl_gput_right:Nn \g_@@_preamble_tl
2010   {
2011     > {

```

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.

```

2012       \dim_set:Nn \l_@@_col_width_dim { #4 }
2013       \hbox_set:Nw \l_@@_cell_box
2014       \@@_cell_begin:w
2015       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2016     }
2017   c
2018   < {
2019     \@@_cell_end:
2020     #1
2021     \hbox_set_end:

```

```

2022         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2023         \@@_adjust_size_box:
2024         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2025     }
2026 }

```

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

```

2027     \int_gincr:N \c@jCol
2028     \@@_patch_preamble_xi:n
2029 }

```

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).

```

2030 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2031 {
2032     \str_if_eq:nnTF { #1 } { [ ]
2033     { \@@_patch_preamble_vii_i:w [ ]
2034     { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2035     }
2036 \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2037 { \@@_patch_preamble_vii_ii:n { #1 } }
2038 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2039 {

```

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

```

2040     \cs_if_exist:NTF \siunitx_cell_begin:w
2041     {
2042         \tl_gput_right:Nn \g_@@_preamble_tl
2043         {
2044             > {
2045                 \@@_cell_begin:w
2046                 \keys_set:nn { siunitx } { #1 }
2047                 \siunitx_cell_begin:w
2048             }
2049             c
2050             < { \siunitx_cell_end: \@@_cell_end: }
2051         }

```

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

```

2052     \int_gincr:N \c@jCol
2053     \@@_patch_preamble_xi:n
2054 }
2055 { \@@_fatal:n { Version~of~siunitx~too~old } }
2056 }

```

For (, [and \{.

```

2057 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2058 {
2059     \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.

```

2060     \int_compare:nNnTF \c@jCol = \c_zero_int
2061     {
2062         \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2063         {

```

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

```

2064         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2065         \tl_gset:Nn \g_@@_right_delim_tl { . }
2066         \@@_patch_preamble:n #2
2067     }
2068 {

```

```

2069         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2070         \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2071     }
2072 }
2073 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2074 }
2075 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2076 {
2077     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2078     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2079     \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
2080     {
2081         \@@_error:nn { delimiter~after~opening } { #2 }
2082         \@@_patch_preamble:n
2083     }
2084     { \@@_patch_preamble:n #2 }
2085 }

```

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}`).

```

2086 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2087 {
2088     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2089     \tl_if_in:nnTF { ) ] \} } { #2 }
2090     { \@@_patch_preamble_ix_i:nnn #1 #2 }
2091     {
2092         \tl_if_eq:nnTF { \q_stop } { #2 }
2093         {
2094             \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2095             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2096             {
2097                 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2098                 \tl_gput_right:Nx \g_@@_internal_code_after_tl
2099                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2100                 \@@_patch_preamble:n #2
2101             }
2102         }
2103         {
2104             \tl_if_in:nnT { ( [ \{ } { #2 }
2105             { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2106             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2107             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2108             \@@_patch_preamble:n #2
2109         }
2110     }
2111 }
2112 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2113 {
2114     \tl_if_eq:nnTF { \q_stop } { #3 }
2115     {
2116         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2117         {
2118             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2119             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2120             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2121             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2122         }
2123         {
2124             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2125             \tl_gput_right:Nx \g_@@_internal_code_after_tl

```



```

2126         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2127         \@@_error:nn { double-closing-delimiter } { #2 }
2128     }
2129 }
2130 {
2131     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2132     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2133     \@@_error:nn { double-closing-delimiter } { #2 }
2134     \@@_patch_preamble:n #3
2135 }
2136 }

```

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.

```

2137 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2138 {
2139     \str_if_eq:nnTF { #1 } { [ ]
2140         { \@@_patch_preamble_x_i:w [ ]
2141             { \@@_patch_preamble_x_i:w [ ] #1 }
2142         }
2143     \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2144         { \@@_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).

```

2145 \keys_define:nn { WithArrows / X-column }
2146 { 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.

```

2147 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2148 {

```

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).

```

2149     \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).

```

2150     \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.

```

2151     \int_zero_new:N \l_@@_weight_int
2152     \int_set:Nn \l_@@_weight_int { 1 }
2153     \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
2154     \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2155     \int_compare:nNnT \l_@@_weight_int < 0
2156     {
2157         \exp_args:Nnx \@@_error:nn { negative-weight }
2158         { \int_use:N \l_@@_weight_int }
2159         \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2160     }
2161     \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).

```

2162     \bool_if:NTF \l_@@_X_columns_aux_bool
2163     {
2164         \@@_patch_preamble_iv_iv:nn

```

```

2165     { \l_@@_weight_int \l_@@_X_columns_dim }
2166     { minipage }
2167   }
2168   {
2169     \tl_gput_right:Nn \g_@@_preamble_tl
2170     {
2171       > {
2172         \@@_cell_begin:w
2173         \bool_set_true:N \l_@@_X_column_bool

```

The following code will nullify the box of the cell.

```

2174     \tl_gput_right:Nn \g_@@_post_action_cell_tl
2175     { \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`.

```

2176     \begin { minipage } { 5 cm } \arraybackslash
2177   }
2178   c
2179   < {
2180     \end { minipage }
2181     \@@_cell_end:
2182   }
2183 }
2184 \int_gincr:N \c@jCol
2185 \@@_patch_preamble_xi:n
2186 }
2187 }

```

```

2188 \cs_new_protected:Npn \@@_patch_preamble_xii:n #1
2189 {
2190   \tl_gput_right:Nn \g_@@_preamble_tl
2191   { ! { \skip_horizontal:N 2\l_@@_radius_dim } }

```

The command `\@@_vdottedline:n` is protected, and, therefore, won't be expanded before writing on `\g_@@_internal_code_after_tl`.

```

2192   \tl_gput_right:Nx \g_@@_internal_code_after_tl
2193   { \@@_vdottedline:n { \int_use:N \c@jCol } }
2194   \@@_patch_preamble:n
2195 }

```

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.

```

2196 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2197 {
2198   \str_if_eq:nnTF { #1 } { < }
2199   \@@_patch_preamble_xiii:n
2200   {
2201     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2202     {
2203       \tl_gput_right:Nn \g_@@_preamble_tl
2204       { ! { \skip_horizontal:N \arrayrulewidth } }
2205     }
2206     {
2207       \exp_args:NNx
2208       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2209       {
2210         \tl_gput_right:Nn \g_@@_preamble_tl
2211         { ! { \skip_horizontal:N \arrayrulewidth } }
2212       }
2213     }
2214     \@@_patch_preamble:n { #1 }
2215   }
2216 }

```

```

2217 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2218 {
2219   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2220   \@@_patch_preamble_xi:n
2221 }

```

The redefinition of `\multicolumn`

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

```

2222 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2223 {

```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```

2224   \multispan { #1 }
2225   \begingroup
2226   \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

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

```

2227   \@temptokena = { #2 }
2228   \@tempswattrue
2229   \@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.

```

2230   \tl_gclear:N \g_@@_preamble_tl
2231   \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

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

```

2232   \exp_args:NV \mkpream \g_@@_preamble_tl
2233   \@addtopreamble \@empty
2234   \endgroup

```

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

```

2235   \int_compare:nNnT { #1 } > 1
2236   {
2237     \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2238     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2239     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2240     \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2241     {
2242       {
2243         \int_compare:nNnTF \c@jCol = 0
2244         { \int_eval:n { \c@iRow + 1 } }
2245         { \int_use:N \c@iRow }
2246       } % modified 2022/01/10
2247       { \int_eval:n { \c@jCol + 1 } }
2248       {
2249         \int_compare:nNnTF \c@jCol = 0
2250         { \int_eval:n { \c@iRow + 1 } }
2251         { \int_use:N \c@iRow }
2252       } % modified 2022/01/10
2253       { \int_eval:n { \c@jCol + #1 } }
2254       { } % for the name of the block
2255     }
2256   }

```

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

```
2257 \cs_set:Npn \@sharp { #3 }
2258 \@arstrut
2259 \@preamble
2260 \null
```

We add some lines.

```
2261 \int_gadd:Nn \c@jCol { #1 - 1 }
2262 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2263 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2264 \ignorespaces
2265 }
```

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`.

```
2266 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2267 {
2268   \str_case:nnF { #1 }
2269   {
2270     c { \@@_patch_m_preamble_i:n #1 }
2271     l { \@@_patch_m_preamble_i:n #1 }
2272     r { \@@_patch_m_preamble_i:n #1 }
2273     > { \@@_patch_m_preamble_ii:nn #1 }
2274     ! { \@@_patch_m_preamble_ii:nn #1 }
2275     @ { \@@_patch_m_preamble_ii:nn #1 }
2276     | { \@@_patch_m_preamble_iii:n #1 }
2277     p { \@@_patch_m_preamble_iv:nnn t #1 }
2278     m { \@@_patch_m_preamble_iv:nnn c #1 }
2279     b { \@@_patch_m_preamble_iv:nnn b #1 }
2280     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2281     \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2282     \q_stop { }
2283   }
2284   { \@@_fatal:nn { unknown~column~type } { #1 } }
2285 }
```

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

```
2286 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2287 {
2288   \tl_gput_right:Nn \g_@@_preamble_tl
2289   {
2290     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2291     #1
2292     < \@@_cell_end:
2293   }
2294 }
```

We test for the presence of a `<`.

```
2294 \@@_patch_m_preamble_x:n
2295 }
```

For `>`, `!` and `@`

```
2296 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2297 {
2298   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2299   \@@_patch_m_preamble:n
2300 }
```

For `|`

```
2301 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2302 {
2303   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2304   \@@_patch_m_preamble:n
2305 }
```

For p, m and b

```

2306 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2307 {
2308   \tl_gput_right:Nn \g_@@_preamble_tl
2309   {
2310     > {
2311       \@@_cell_begin:w
2312       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2313       \mode_leave_vertical:
2314       \arraybackslash
2315       \vrule height \box_ht:N \@@arstrutbox depth 0 pt width 0 pt
2316     }
2317     c
2318     < {
2319       \vrule height 0 pt depth \box_dp:N \@@arstrutbox width 0 pt
2320       \end { minipage }
2321       \@@_cell_end:
2322     }
2323   }

```

We test for the presence of a <.

```

2324   \@@_patch_m_preamble_x:n
2325 }

```

For w and W

```

2326 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2327 {
2328   \tl_gput_right:Nn \g_@@_preamble_tl
2329   {
2330     > {
2331       \hbox_set:Nw \l_@@_cell_box
2332       \@@_cell_begin:w
2333       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2334     }
2335     c
2336     < {
2337       \@@_cell_end:
2338       #1
2339       \hbox_set_end:
2340       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2341       \@@_adjust_size_box:
2342       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2343     }
2344   }

```

We test for the presence of a <.

```

2345   \@@_patch_m_preamble_x:n
2346 }

```

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.

```

2347 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2348 {
2349   \str_if_eq:nnTF { #1 } { < }
2350   \@@_patch_m_preamble_ix:n
2351   {
2352     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2353     {
2354       \tl_gput_right:Nn \g_@@_preamble_tl
2355       { ! { \skip_horizontal:N \arrayrulewidth } }
2356     }
2357     {
2358       \exp_args:NNx
2359       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }

```

```

2360         {
2361             \tl_gput_right:Nn \g_@@_preamble_tl
2362             { ! { \skip_horizontal:N \arrayrulewidth } }
2363         }
2364     }
2365     \@@_patch_m_preamble:n { #1 }
2366 }
2367 }
2368 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2369 {
2370     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2371     \@@_patch_m_preamble_x:n
2372 }

```

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`).

```

2373 \cs_new_protected:Npn \@@_put_box_in_flow:
2374 {
2375     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2376     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2377     \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2378     { \box_use_drop:N \l_tmpa_box }
2379     \@@_put_box_in_flow_i:
2380 }

```

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).

```

2381 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2382 {
2383     \pgfpicture
2384     \@@_qpoint:n { row - 1 }
2385     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2386     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2387     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2388     \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).

```

2389     \str_if_in:NnTF \l_@@_baseline_tl { line- }
2390     {
2391         \int_set:Nn \l_tmpa_int
2392         {
2393             \str_range:Nnn
2394             \l_@@_baseline_tl
2395             6
2396             { \tl_count:V \l_@@_baseline_tl }
2397         }
2398         \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2399     }
2400     {
2401         \str_case:VnF \l_@@_baseline_tl
2402         {
2403             { t } { \int_set:Nn \l_tmpa_int 1 }
2404             { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2405         }
2406         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2407         \bool_lazy_or:nnT
2408         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2409         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2410         {

```

```

2411         \@@_error:n { bad-value-for-baseline }
2412         \int_set:Nn \l_tmpa_int 1
2413     }
2414     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2415         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2416     }
2417     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2418     \endpgfpicture
2419     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2420     \box_use_drop:N \l_tmpa_box
2421 }

```

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).

```

2422 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2423 {

```

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.

```

2424     \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2425     {
2426         \box_set_wd:Nn \l_@@_the_array_box
2427         { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2428     }

```

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

```

2429     \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.

```

2430     \hbox
2431     {
2432         \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.

```

2433         \@@_create_extra_nodes:
2434         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2435     }
2436     \bool_lazy_or:nnT
2437     { \int_compare_p:nNn \c@tabularnote > 0 }
2438     { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2439     \@@_insert_tabularnotes:
2440     \end { minipage }
2441 }
2442 \cs_new_protected:Npn \@@_insert_tabularnotes:
2443 {
2444     \skip_vertical:N 0.65ex

```

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

```

2445     \group_begin:
2446     \l_@@_notes_code_before_tl
2447     \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.

```

2448 \int_compare:nNnT \c@tabularnote > 0
2449 {
2450   \bool_if:NTF \l_@@_notes_para_bool
2451   {
2452     \begin { tabularnotes* }
2453     \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2454     \end { tabularnotes* }

```

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

```

2455     \par
2456   }
2457 {
2458   \tabularnotes
2459   \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2460   \endtabularnotes
2461 }
2462 }
2463 \unskip
2464 \group_end:
2465 \bool_if:NT \l_@@_notes_bottomrule_bool
2466 {
2467   \bool_if:NTF \c_@@_booktabs_loaded_bool
2468   {

```

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

```

2469     \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2470     { \CT@arc@ \hrule height \heavyrulewidth }
2471   }
2472   { @@_error:n { bottomrule~without~booktabs } }
2473 }
2474 \l_@@_notes_code_after_tl
2475 \seq_gclear:N \g_@@_tabularnotes_seq
2476 \int_gzero:N \c@tabularnote
2477 }

```

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`.

```

2478 \cs_new_protected:Npn @@_use_arraybox_with_notes_b:
2479 {
2480   \pgfpicture
2481   \@@_qpoint:n { row - 1 }
2482   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2483   \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2484   \dim_gsub:Nn \g_tmpa_dim \pgf@y
2485   \endpgfpicture
2486   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2487   \int_compare:nNnT \l_@@_first_row_int = 0
2488   {
2489     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2490     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2491   }
2492   \box_move_up:nn \g_tmpa_dim { \hbox { @@_use_arraybox_with_notes_c: } }
2493 }

```

Now, the general case.

```

2494 \cs_new_protected:Npn @@_use_arraybox_with_notes:
2495 {

```


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

```
2496 \tl_if_eq:NnT \l_@@_baseline_tl { t }
2497 { \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`.

```
2498 \pgfpicture
2499 \@@_qpoint:n { row - 1 }
2500 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2501 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2502 {
2503   \int_set:Nn \l_tmpa_int
2504   {
2505     \str_range:Nnn
2506     \l_@@_baseline_tl
2507     6
2508     { \tl_count:V \l_@@_baseline_tl }
2509   }
2510   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2511 }
2512 {
2513   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2514   \bool_lazy_or:nnT
2515   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2516   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2517   {
2518     \@@_error:n { bad~value~for~baseline }
2519     \int_set:Nn \l_tmpa_int 1
2520   }
2521   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2522 }
2523 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2524 \endpgfpicture
2525 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2526 \int_compare:nNnT \l_@@_first_row_int = 0
2527 {
2528   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2529   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2530 }
2531 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2532 }
```

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.

```
2533 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2534 {
```

We will compute the real width of both delimiters used.

```
2535 \dim_zero_new:N \l_@@_real_left_delim_dim
2536 \dim_zero_new:N \l_@@_real_right_delim_dim
2537 \hbox_set:Nn \l_tmpb_box
2538 {
2539   \c_math_toggle_token
2540   \left #1
2541   \vcenter
2542   {
2543     \vbox_to_ht:nn
2544     { \box_ht_plus_dp:N \l_tmpa_box }
2545     { }
2546   }
2547   \right .
2548   \c_math_toggle_token
```

```

2549     }
2550     \dim_set:Nn \l_@@_real_left_delim_dim
2551     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2552     \hbox_set:Nn \l_tmpb_box
2553     {
2554       \c_math_toggle_token
2555       \left .
2556       \vbox_to_ht:nn
2557         { \box_ht_plus_dp:N \l_tmpa_box }
2558         { }
2559       \right #2
2560       \c_math_toggle_token
2561     }
2562     \dim_set:Nn \l_@@_real_right_delim_dim
2563     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2564     \skip_horizontal:N \l_@@_left_delim_dim
2565     \skip_horizontal:N -\l_@@_real_left_delim_dim
2566     \@@_put_box_in_flow:
2567     \skip_horizontal:N \l_@@_right_delim_dim
2568     \skip_horizontal:N -\l_@@_real_right_delim_dim
2569   }

```

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).

```

2570 \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).

```

2571 {
2572   \peek_meaning_ignore_spaces:NTF \end \@@_analyze_end:Nn

```

Here is the call to `\array` (we have a dedicated macro `\@@_array:` because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

2573     { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2574   }
2575   {
2576     \@@_create_col_nodes:
2577     \endarray
2578   }

```

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`).

```

2579 \NewDocumentEnvironment { @@-light-syntax } { b }
2580 {

```

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.

```

2581   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2582   \tl_map_inline:nn { #1 }
2583   {
2584     \str_if_eq:nnT { ##1 } { & }
2585     { \@@_fatal:n { ampersand-in-light-syntax } }
2586     \str_if_eq:nnT { ##1 } { \ }
2587     { \@@_fatal:n { double-backslash-in-light-syntax } }
2588   }

```

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`.

```
2589   \@@_light_syntax_i #1 \CodeAfter \q_stop
2590 }
```

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`.

```
2591 { }
2592 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2593 {
2594   \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).

```
2595   \seq_gclear_new:N \g_@@_rows_seq
2596   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2597   \exp_args:NNV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to know that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```
2598   \int_compare:nNnT \l_@@_last_row_int = { -1 }
2599     { \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`).

```
2600   \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.

```
2601   \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2602   \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
2603   \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2604   \@@_create_col_nodes:
2605   \endarray
2606 }
2607 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2608 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
2609 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2610 {
2611   \seq_gclear_new:N \g_@@_cells_seq
2612   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2613   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2614   \l_tmpa_tl
2615   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2616 }
```

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).

```
2617 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2618 {
2619   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2620     { \@@_fatal:n { empty-environment } } }
```

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

```
2621   \end { #2 }
2622 }
```

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).

```

2623 \cs_new:Npn \@@_create_col_nodes:
2624 {
2625   \crrc
2626   \int_compare:nNnT \l_@@_first_col_int = 0
2627   {
2628     \omit
2629     \hbox_overlap_left:n
2630     {
2631       \bool_if:NT \l_@@_code_before_bool
2632       { \pgfsys@markposition { \@@_env: - col - 0 } }
2633       \pgfpicture
2634       \pgfrememberpicturerepositiononpagetrue
2635       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
2636       \str_if_empty:NF \l_@@_name_str
2637       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2638       \endpgfpicture
2639       \skip_horizontal:N 2\col@sep
2640       \skip_horizontal:N \g_@@_width_first_col_dim
2641     }
2642     &
2643   }
2644   \omit

```

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

```

2645   \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`).

```

2646   \int_compare:nNnTF \l_@@_first_col_int = 0
2647   {
2648     \bool_if:NT \l_@@_code_before_bool
2649     {
2650       \hbox
2651       {
2652         \skip_horizontal:N -0.5\arrayrulewidth
2653         \pgfsys@markposition { \@@_env: - col - 1 }
2654         \skip_horizontal:N 0.5\arrayrulewidth
2655       }
2656     }
2657     \pgfpicture
2658     \pgfrememberpicturerepositiononpagetrue
2659     \pgfcoordinate { \@@_env: - col - 1 }
2660     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
2661     \str_if_empty:NF \l_@@_name_str
2662     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2663     \endpgfpicture
2664   }
2665   {
2666     \bool_if:NT \l_@@_code_before_bool
2667     {
2668       \hbox
2669       {
2670         \skip_horizontal:N 0.5\arrayrulewidth
2671         \pgfsys@markposition { \@@_env: - col - 1 }
2672         \skip_horizontal:N -0.5\arrayrulewidth
2673       }
2674     }
2675     \pgfpicture
2676     \pgfrememberpicturerepositiononpagetrue
2677     \pgfcoordinate { \@@_env: - col - 1 }
2678     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }

```

```

2679 \str_if_empty:NF \l_@@_name_str
2680 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2681 \endpgfpicture
2682 }

```

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.

```

2683 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2684 \bool_if:NF \l_@@_auto_columns_width_bool
2685 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2686 {
2687   \bool_lazy_and:nnTF
2688     \l_@@_auto_columns_width_bool
2689     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2690     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2691     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2692   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2693 }
2694 \skip_horizontal:N \g_tmpa_skip
2695 \hbox
2696 {
2697   \bool_if:NT \l_@@_code_before_bool
2698   {
2699     \hbox
2700     {
2701       \skip_horizontal:N -0.5\arrayrulewidth
2702       \pgfsys@markposition { \@@_env: - col - 2 }
2703       \skip_horizontal:N 0.5\arrayrulewidth
2704     }
2705   }
2706   \pgfpicture
2707   \pgfrememberpicturepositiononpagetrue
2708   \pgfcoordinate { \@@_env: - col - 2 }
2709   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2710   \str_if_empty:NF \l_@@_name_str
2711   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2712   \endpgfpicture
2713 }

```

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.

```

2714 \int_gset:Nn \g_tmpa_int 1
2715 \bool_if:NTF \g_@@_last_col_found_bool
2716 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2717 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2718 {
2719   &
2720   \omit
2721   \int_gincr:N \g_tmpa_int

```

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

```

2722 \skip_horizontal:N \g_tmpa_skip
2723 \bool_if:NT \l_@@_code_before_bool
2724 {
2725   \hbox
2726   {
2727     \skip_horizontal:N -0.5\arrayrulewidth
2728     \pgfsys@markposition
2729     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2730     \skip_horizontal:N 0.5\arrayrulewidth

```

```

2731     }
2732   }

```

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

```

2733   \pgfpicture
2734     \pgfrememberpicturepositiononpagetrue
2735     \pgfcoordinate { \l_@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2736     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2737     \str_if_empty:NF \l_@@_name_str
2738     {
2739       \pgfnodealias
2740       { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
2741       { \l_@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2742     }
2743   \endpgfpicture
2744 }

```

```

2745   &
2746   \omit

```

The two following lines have been added on 2021-12-15 to solve a bug mentioned by Joao Luis Soares by mail.

```

2747   \int_compare:nNnT \g_@@_col_total_int = 1
2748   { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
2749   \skip_horizontal:N \g_tmpa_skip
2750   \int_gincr:N \g_tmpa_int
2751   \bool_lazy_all:nT
2752   {
2753     \l_@@_NiceArray_bool
2754     { \bool_not_p:n \l_@@_NiceTabular_bool }
2755     { \clist_if_empty_p:N \l_@@_vlines_clist }
2756     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2757     { ! \l_@@_bar_at_end_of_pream_bool }
2758   }
2759   { \skip_horizontal:N -\col@sep }
2760   \bool_if:NT \l_@@_code_before_bool
2761   {
2762     \hbox
2763     {
2764       \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.

```

2765       \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2766       { \skip_horizontal:N -\arraycolsep }
2767       \pgfsys@markposition
2768       { \l_@@_env: - col - \int_eval:n {
2769         \g_tmpa_int + 1 } }
2770       \skip_horizontal:N 0.5\arrayrulewidth
2771       \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2772       { \skip_horizontal:N \arraycolsep }
2773     }
2774   }
2775   \pgfpicture
2776     \pgfrememberpicturepositiononpagetrue
2777     \pgfcoordinate { \l_@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2778     {
2779       \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
2780       {
2781         \pgfpoint
2782         { - 0.5 \arrayrulewidth - \arraycolsep }
2783         \c_zero_dim
2784       }

```

```

2785         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2786     }
2787     \str_if_empty:NF \l_@@_name_str
2788     {
2789         \pgfnodealias
2790         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
2791         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2792     }
2793     \endpgfpicture

2794     \bool_if:NT \g_@@_last_col_found_bool
2795     {
2796         \hbox_overlap_right:n
2797         {
2798             \skip_horizontal:N \g_@@_width_last_col_dim
2799             \bool_if:NT \l_@@_code_before_bool
2800             {
2801                 \pgfsys@markposition
2802                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2803             }
2804             \pgfpicture
2805             \pgfrememberpicturerepositiononpagetrue
2806             \pgfcoordinate
2807             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2808             \pgfpointorigin
2809             \str_if_empty:NF \l_@@_name_str
2810             {
2811                 \pgfnodealias
2812                 {
2813                     \l_@@_name_str - col
2814                     - \int_eval:n { \g_@@_col_total_int + 1 }
2815                 }
2816                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2817             }
2818             \endpgfpicture
2819         }
2820     }
2821     \cr
2822 }

```

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

```

2823 \tl_const:Nn \c_@@_preamble_first_col_tl
2824 {
2825     >
2826     {

```

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).

```

2827     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
2828     \bool_gset_true:N \g_@@_after_col_zero_bool
2829     \@@_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.

```

2830     \hbox_set:Nw \l_@@_cell_box
2831     \@@_math_toggle_token:
2832     \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”.

```

2833     \bool_lazy_and:nnT
2834     { \int_compare_p:nNn \c@iRow > 0 }

```

```

2835     {
2836         \bool_lazy_or_p:nn
2837         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2838         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2839     }
2840     {
2841         \l_@@_code_for_first_col_tl
2842         \xglobal \colorlet { nicematrix-first-col } { . }
2843     }
2844 }

```

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`.

```

2845     l
2846     <
2847     {
2848         \@@_math_toggle_token:
2849         \hbox_set_end:
2850         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2851         \@@_adjust_size_box:
2852         \@@_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.

```

2853         \dim_gset:Nn \g_@@_width_first_col_dim
2854         { \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.

```

2855         \hbox_overlap_left:n
2856         {
2857             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2858             \@@_node_for_cell:
2859             { \box_use_drop:N \l_@@_cell_box }
2860             \skip_horizontal:N \l_@@_left_delim_dim
2861             \skip_horizontal:N \l_@@_left_margin_dim
2862             \skip_horizontal:N \l_@@_extra_left_margin_dim
2863         }
2864         \bool_gset_false:N \g_@@_empty_cell_bool
2865         \skip_horizontal:N -2\col@sep
2866     }
2867 }

```

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

```

2868 \tl_const:Nn \c_@@_preamble_last_col_tl
2869 {
2870     >
2871     {

```

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).

```

2872         \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.

```

2873         \bool_gset_true:N \g_@@_last_col_found_bool
2874         \int_gincr:N \c@jCol
2875         \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.

```

2876         \hbox_set:Nw \l_@@_cell_box
2877         \@@_math_toggle_token:
2878         \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”.

```

2879     \int_compare:nNnT \c@iRow > 0
2880     {
2881         \bool_lazy_or:nnT
2882         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2883         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2884         {
2885             \l_@@_code_for_last_col_tl
2886             \xglobal \colorlet { nicematrix-last-col } { . }
2887         }
2888     }
2889 }
2890
2891 <
2892 {
2893     \@@_math_toggle_token:
2894     \hbox_set_end:
2895     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2896     \@@_adjust_size_box:
2897     \@@_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.

```

2898     \dim_gset:Nn \g_@@_width_last_col_dim
2899     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2900     \skip_horizontal:N -2\col@sep

```

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

```

2901     \hbox_overlap_right:n
2902     {
2903         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2904         {
2905             \skip_horizontal:N \l_@@_right_delim_dim
2906             \skip_horizontal:N \l_@@_right_margin_dim
2907             \skip_horizontal:N \l_@@_extra_right_margin_dim
2908             \@@_node_for_cell:
2909         }
2910     }
2911     \bool_gset_false:N \g_@@_empty_cell_bool
2912 }
2913 }

```

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.

```

2914 \NewDocumentEnvironment { NiceArray } { }
2915 {
2916     \bool_set_true:N \l_@@_NiceArray_bool
2917     \str_if_empty:NT \g_@@_name_env_str
2918     { \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).

```

2919     \NiceArrayWithDelims . .
2920 }
2921 { \endNiceArrayWithDelims }

```

We create the variants of the environment `{NiceArrayWithDelims}`.

```

2922 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2923 {

```

```

2924 \NewDocumentEnvironment { #1 NiceArray } { }
2925 {
2926   \str_if_empty:NT \g_@@_name_env_str
2927   { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2928   \@@_test_if_math_mode:
2929   \NiceArrayWithDelims #2 #3
2930 }
2931 { \endNiceArrayWithDelims }
2932 }
2933 \@@_def_env:nnn p ( )
2934 \@@_def_env:nnn b [ ]
2935 \@@_def_env:nnn B \{ \}
2936 \@@_def_env:nnn v | |
2937 \@@_def_env:nnn V \| \|

```

The environment `{NiceMatrix}` and its variants

```

2938 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2939 {
2940   \bool_set_true:N \l_@@_Matrix_bool
2941   \use:c { #1 NiceArray }
2942   {
2943     *
2944     {
2945       \int_compare:nNnTF \l_@@_last_col_int < 0
2946       \c@MaxMatrixCols
2947       { \int_eval:n { \l_@@_last_col_int - 1 } }
2948     }
2949     { > \@@_cell_begin:w #2 < \@@_cell_end: }
2950   }
2951 }
2952 \clist_map_inline:nn { { } , p , b , B , v , V }
2953 {
2954   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2955   {
2956     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2957     \tl_set:Nn \l_@@_type_of_col_tl c
2958     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2959     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2960   }
2961   { \use:c { end #1 NiceArray } }
2962 }

```

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

```

2963 \cs_new_protected:Npn \@@_NotEmpty:
2964 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

2965 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2966 {

```

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`.

```

2967   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
2968   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
2969   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2970   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2971   \bool_set_true:N \l_@@_NiceTabular_bool
2972   \NiceArray { #2 }

```

```

2973 }
2974 { \endNiceArray }

2975 \cs_set_protected:Npn \@@_newcolumnntype #1
2976 {
2977   \cs_if_free:cT { NC @ find @ #1 }
2978   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
2979   \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
2980   \peek_meaning:NTF [
2981     { \newcol@ #1 }
2982     { \newcol@ #1 [ 0 ] }
2983   }

2984 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
2985 {

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}).
2986   \bool_if:NT \c_@@_tabularx_loaded_bool
2987     { \newcolumnntype { X } { \@@_X } }
2988   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
2989   \dim_zero_new:N \l_@@_width_dim
2990   \dim_set:Nn \l_@@_width_dim { #1 }
2991   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2992   \bool_set_true:N \l_@@_NiceTabular_bool
2993   \NiceArray { #3 }
2994 }
2995 { \endNiceArray }

2996 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
2997 {
2998   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2999   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3000   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3001   \bool_set_true:N \l_@@_NiceTabular_bool
3002   \NiceArray { #3 }
3003 }
3004 { \endNiceArray }

```

After the construction of the array

```

3005 \cs_new_protected:Npn \@@_after_array:
3006 {
3007   \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.

```

3008   \bool_if:NT \g_@@_last_col_found_bool
3009     { \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`.

```

3010   \bool_if:NT \l_@@_last_col_without_value_bool
3011     { \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.

```

3012   \bool_if:NT \l_@@_last_row_without_value_bool
3013     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3014 \tl_gput_right:Nx \g_@@_aux_tl
3015 {
3016   \seq_gset_from_clist:Nn \exp_not:N \c_@@_size_seq
3017   {
3018     \int_use:N \l_@@_first_row_int ,
3019     \int_use:N \c@iRow ,
3020     \int_use:N \g_@@_row_total_int ,
3021     \int_use:N \l_@@_first_col_int ,
3022     \int_use:N \c@jCol ,
3023     \int_use:N \g_@@_col_total_int
3024   }
3025 }

```

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`).

```

3026 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3027 {
3028   \tl_gput_right:Nx \g_@@_aux_tl
3029   {
3030     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3031     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3032   }
3033 }
3034 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3035 {
3036   \tl_gput_right:Nx \g_@@_aux_tl
3037   {
3038     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3039     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3040     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3041     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3042   }
3043 }

```

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

```

3044 \@@_create_diag_nodes:

```

We create the aliases using `last` for the nodes of the cells in the last row and the last column.

```

3045 \pgfpicture
3046 \int_step_inline:nn \c@iRow
3047 {
3048   \pgfnodealias
3049   { \@@_env: - ##1 - last }
3050   { \@@_env: - ##1 - \int_use:N \c@jCol }
3051 }
3052 \int_step_inline:nn \c@jCol
3053 {
3054   \pgfnodealias
3055   { \@@_env: - last - ##1 }
3056   { \@@_env: - \int_use:N \c@iRow - ##1 }
3057 }
3058 \str_if_empty:NF \l_@@_name_str
3059 {
3060   \int_step_inline:nn \c@iRow
3061   {
3062     \pgfnodealias
3063     { \l_@@_name_str - ##1 - last }
3064     { \@@_env: - ##1 - \int_use:N \c@jCol }
3065   }
3066   \int_step_inline:nn \c@jCol
3067   {
3068     \pgfnodealias
3069     { \l_@@_name_str - last - ##1 }

```

```

3070         { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3071     }
3072 }
3073 \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.

```

3074 \bool_if:NT \l_@@_parallelize_diags_bool
3075 {
3076     \int_gzero_new:N \g_@@_ddots_int
3077     \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```

3078     \dim_gzero_new:N \g_@@_delta_x_one_dim
3079     \dim_gzero_new:N \g_@@_delta_y_one_dim
3080     \dim_gzero_new:N \g_@@_delta_x_two_dim
3081     \dim_gzero_new:N \g_@@_delta_y_two_dim
3082 }
3083 \int_zero_new:N \l_@@_initial_i_int
3084 \int_zero_new:N \l_@@_initial_j_int
3085 \int_zero_new:N \l_@@_final_i_int
3086 \int_zero_new:N \l_@@_final_j_int
3087 \bool_set_false:N \l_@@_initial_open_bool
3088 \bool_set_false:N \l_@@_final_open_bool

```

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

```

3089 \bool_if:NT \l_@@_small_bool
3090 {
3091     \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
3092     \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }

```

The dimension `\l_@@_xdots_shorten_dim` corresponds to the option `xdots/shorten` available to the user. That's why we give a new value according to the current value, and not an absolute value.

```

3093     \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
3094 }

```

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

```

3095 \l_@@_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.

```

3096 \l_@@_compute_corners:

```

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

```

3097 \l_@@_adjust_pos_of_blocks_seq:
3098 \tl_if_empty:NF \l_@@_hlines_clist \l_@@_draw_hlines:
3099 \tl_if_empty:NF \l_@@_vlines_clist \l_@@_draw_vlines:

```

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

```

3100 \bool_if:NT \c_@@_tikz_loaded_bool
3101 {
3102     \tikzset

```

⁶⁸It's possible to use the option `parallelize-diags` to disable this parallelization.

```

3103     {
3104         every~picture / .style =
3105         {
3106             overlay ,
3107             remember~picture ,
3108             name~prefix = \@@_env: -
3109         }
3110     }
3111 }
3112 \cs_set_eq:NN \ialign \@@_old_ialign:
3113 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3114 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3115 \cs_set_eq:NN \OverBrace \@@_OverBrace
3116 \cs_set_eq:NN \line \@@_line
3117 \g_@@_internal_code_after_tl
3118 \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.

```

3119 \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).

```

3120 \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:`.

```

3121 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3122 \scan_stop:
3123 \tl_gclear:N \g_nicematrix_code_after_tl
3124 \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.

```

3125 \tl_if_empty:NF \g_nicematrix_code_before_tl
3126 {

```

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`.

```

3127 \cs_set_protected:Npn \rectanglecolor { }
3128 \cs_set_protected:Npn \columncolor { }
3129 \tl_gput_right:Nx \g_@@_aux_tl
3130 {
3131     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3132     { \exp_not:V \g_nicematrix_code_before_tl }
3133 }
3134 \bool_set_true:N \l_@@_code_before_bool
3135 }

3136 \str_gclear:N \g_@@_name_env_str
3137 \@@_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`.

```

3138 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@

```

⁶⁹e.g. `\color[rgb]{0.5,0.5,0}`

```
3139 }
```

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`.

```
3140 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3141 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }
```

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

```
3142 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3143 {
3144   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3145   { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3146 }
```

The following command must *not* be protected.

```
3147 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3148 {
3149   { #1 }
3150   { #2 }
3151   {
3152     \int_compare:nNnTF { #3 } > { 99 }
3153     { \int_use:N \c@iRow }
3154     { #3 }
3155   }
3156   {
3157     \int_compare:nNnTF { #4 } > { 99 }
3158     { \int_use:N \c@jCol }
3159     { #4 }
3160   }
3161   { #5 }
3162 }
```

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).

```
3163 \hook_gput_code:nnn { begindocument } { . }
3164 {
3165   \cs_new_protected:Npx \@@_draw_dotted_lines:
3166   {
3167     \c_@@_pgfortikzpicture_tl
3168     \@@_draw_dotted_lines_i:
3169     \c_@@_endpgfortikzpicture_tl
3170   }
3171 }
```

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

```
3172 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3173 {
3174   \pgfrememberpicturepositiononpagetrue
3175   \pgf@relevantforpicturesizefalse
3176   \g_@@_HVdotsfor_lines_tl
3177   \g_@@_Vdots_lines_tl
3178   \g_@@_Ddots_lines_tl
3179   \g_@@_Iddots_lines_tl
3180   \g_@@_Cdots_lines_tl
```

```

3181 \g_@@_Ldots_lines_tl
3182 }

3183 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3184 {
3185   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3186   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3187 }

```

We define a new PGF shape for the diag nodes because we want to provide a anchor called .5 for those nodes.

```

3188 \pgfdeclareshape { @@_diag_node }
3189 {
3190   \savedanchor { \five }
3191   {
3192     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3193     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3194   }
3195   \anchor { 5 } { \five }
3196   \anchor { center } { \pgfpointorigin }
3197 }

```

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).

```

3198 \cs_new_protected:Npn \@@_create_diag_nodes:
3199 {
3200   \pgfpicture
3201   \pgfrememberpicturepositiononpagetrue
3202   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3203   {
3204     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3205     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3206     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3207     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3208     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3209     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3210     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3211     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3212     \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.

```

3213     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3214     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3215     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3216     \str_if_empty:NF \l_@@_name_str
3217     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3218   }

```

Now, the last node. Of course, that is only a coordinate because there is not .5 anchor for that node.

```

3219   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3220   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3221   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3222   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3223   \pgfcoordinate
3224   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3225   \pgfnodealias
3226   { \@@_env: - last }
3227   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3228   \str_if_empty:NF \l_@@_name_str
3229   {
3230     \pgfnodealias

```



```

3231     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3232     { \@@_env: - \int_use:N \l_tmpa_int }
3233     \pgfnodealias
3234     { \l_@@_name_str - last }
3235     { \@@_env: - last }
3236   }
3237   \endpgfpicture
3238 }

```

We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

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

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```

3239 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3240 {

```

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

```

3241   \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

3242   \int_set:Nn \l_@@_initial_i_int { #1 }
3243   \int_set:Nn \l_@@_initial_j_int { #2 }
3244   \int_set:Nn \l_@@_final_i_int { #1 }
3245   \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.

```

3246   \bool_set_false:N \l_@@_stop_loop_bool
3247   \bool_do_until:Nn \l_@@_stop_loop_bool
3248   {
3249     \int_add:Nn \l_@@_final_i_int { #3 }
3250     \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3251 \bool_set_false:N \l_@@_final_open_bool
3252 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3253 {
3254   \int_compare:nNnTF { #3 } = 1
3255   { \bool_set_true:N \l_@@_final_open_bool }
3256   {
3257     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3258     { \bool_set_true:N \l_@@_final_open_bool }
3259   }
3260 }
3261 {
3262   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3263   {
3264     \int_compare:nNnT { #4 } = { -1 }
3265     { \bool_set_true:N \l_@@_final_open_bool }
3266   }
3267   {
3268     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3269     {
3270       \int_compare:nNnT { #4 } = 1
3271       { \bool_set_true:N \l_@@_final_open_bool }
3272     }
3273   }
3274 }
3275 \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.

```

3276 {

```

We do a step backwards.

```

3277 \int_sub:Nn \l_@@_final_i_int { #3 }
3278 \int_sub:Nn \l_@@_final_j_int { #4 }
3279 \bool_set_true:N \l_@@_stop_loop_bool
3280 }

```

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`.

```

3281 {
3282   \cs_if_exist:cTF
3283   {
3284     @@ _ dotted _
3285     \int_use:N \l_@@_final_i_int -
3286     \int_use:N \l_@@_final_j_int
3287   }
3288   {
3289     \int_sub:Nn \l_@@_final_i_int { #3 }
3290     \int_sub:Nn \l_@@_final_j_int { #4 }
3291     \bool_set_true:N \l_@@_final_open_bool
3292     \bool_set_true:N \l_@@_stop_loop_bool
3293   }
3294   {
3295     \cs_if_exist:cTF
3296     {
3297       pgf @ sh @ ns @ \@@_env:
3298       - \int_use:N \l_@@_final_i_int
3299       - \int_use:N \l_@@_final_j_int
3300     }
3301     { \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.

```

3302         {
3303             \cs_set:cpn
3304             {
3305                 @@ _ dotted _
3306                 \int_use:N \l_@@_final_i_int -
3307                 \int_use:N \l_@@_final_j_int
3308             }
3309             { }
3310         }
3311     }
3312 }
3313 }

```

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

```

3314 \bool_set_false:N \l_@@_stop_loop_bool
3315 \bool_do_until:Nn \l_@@_stop_loop_bool
3316 {
3317     \int_sub:Nn \l_@@_initial_i_int { #3 }
3318     \int_sub:Nn \l_@@_initial_j_int { #4 }
3319     \bool_set_false:N \l_@@_initial_open_bool
3320     \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3321     {
3322         \int_compare:nNnTF { #3 } = 1
3323         { \bool_set_true:N \l_@@_initial_open_bool }
3324         {
3325             \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3326             { \bool_set_true:N \l_@@_initial_open_bool }
3327         }
3328     }
3329     {
3330         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3331         {
3332             \int_compare:nNnT { #4 } = 1
3333             { \bool_set_true:N \l_@@_initial_open_bool }
3334         }
3335         {
3336             \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3337             {
3338                 \int_compare:nNnT { #4 } = { -1 }
3339                 { \bool_set_true:N \l_@@_initial_open_bool }
3340             }
3341         }
3342     }
3343     \bool_if:Ntf \l_@@_initial_open_bool
3344     {
3345         \int_add:Nn \l_@@_initial_i_int { #3 }
3346         \int_add:Nn \l_@@_initial_j_int { #4 }
3347         \bool_set_true:N \l_@@_stop_loop_bool
3348     }
3349     {
3350         \cs_if_exist:cTF
3351         {
3352             @@ _ dotted _
3353             \int_use:N \l_@@_initial_i_int -
3354             \int_use:N \l_@@_initial_j_int
3355         }
3356         {
3357             \int_add:Nn \l_@@_initial_i_int { #3 }
3358             \int_add:Nn \l_@@_initial_j_int { #4 }
3359             \bool_set_true:N \l_@@_initial_open_bool
3360             \bool_set_true:N \l_@@_stop_loop_bool

```

```

3361     }
3362     {
3363         \cs_if_exist:cTF
3364         {
3365             pgf @ sh @ ns @ \l_@@_env:
3366             - \int_use:N \l_@@_initial_i_int
3367             - \int_use:N \l_@@_initial_j_int
3368         }
3369         { \bool_set_true:N \l_@@_stop_loop_bool }
3370         {
3371             \cs_set:cpn
3372             {
3373                 @@ _ dotted _
3374                 \int_use:N \l_@@_initial_i_int -
3375                 \int_use:N \l_@@_initial_j_int
3376             }
3377             { }
3378         }
3379     }
3380 }
3381 }

```

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.

```

3382     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3383     {
3384         { \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`.

```

3385         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3386         { \int_use:N \l_@@_final_i_int }
3387         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3388         { } % for the name of the block
3389     }
3390 }

```

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).

```

3391 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3392 {
3393     \int_set:Nn \l_@@_row_min_int 1
3394     \int_set:Nn \l_@@_col_min_int 1
3395     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3396     \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`.

```

3397     \seq_map_inline:Nn \g_@@_submatrix_seq
3398     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
3399 }

```

#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.

```

3400 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3401 {
3402     \bool_if:nT
3403     {
3404         \int_compare_p:n { #3 <= #1 }
3405         && \int_compare_p:n { #1 <= #5 }
3406         && \int_compare_p:n { #4 <= #2 }

```

```

3407     && \int_compare_p:n { #2 <= #6 }
3408 }
3409 {
3410     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3411     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3412     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3413     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3414 }
3415 }

3416 \cs_new_protected:Npn \@@_set_initial_coords:
3417 {
3418     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3419     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3420 }
3421 \cs_new_protected:Npn \@@_set_final_coords:
3422 {
3423     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3424     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3425 }
3426 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3427 {
3428     \pgfpointanchor
3429     {
3430         \@@_env:
3431         - \int_use:N \l_@@_initial_i_int
3432         - \int_use:N \l_@@_initial_j_int
3433     }
3434     { #1 }
3435     \@@_set_initial_coords:
3436 }
3437 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3438 {
3439     \pgfpointanchor
3440     {
3441         \@@_env:
3442         - \int_use:N \l_@@_final_i_int
3443         - \int_use:N \l_@@_final_j_int
3444     }
3445     { #1 }
3446     \@@_set_final_coords:
3447 }

3448 \cs_new_protected:Npn \@@_open_x_initial_dim:
3449 {
3450     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3451     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3452     {
3453         \cs_if_exist:cT
3454         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3455         {
3456             \pgfpointanchor
3457             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3458             { west }
3459             \dim_set:Nn \l_@@_x_initial_dim
3460             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3461         }
3462     }

```

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

```

3463     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3464     {
3465         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3466         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

3467     \dim_add:Nn \l_@@_x_initial_dim \col@sep
3468   }
3469 }

3470 \cs_new_protected:Npn \@@_open_x_final_dim:
3471 {
3472   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3473   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3474   {
3475     \cs_if_exist:cT
3476     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3477     {
3478       \pgfpointanchor
3479       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3480       { east }
3481       \dim_set:Nn \l_@@_x_final_dim
3482       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3483     }
3484   }

```

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

```

3485   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3486   {
3487     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
3488     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3489     \dim_sub:Nn \l_@@_x_final_dim \col@sep
3490   }
3491 }

```

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.

```

3492 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3493 {
3494   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3495   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3496   {
3497     \@@_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.

```

3498   \group_begin:
3499   \int_compare:nNnTF { #1 } = 0
3500   { \color { nicematrix-first-row } }
3501   {

```

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.

```

3502     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3503     { \color { nicematrix-last-row } }
3504   }
3505   \keys_set:nn { NiceMatrix / xdots } { #3 }
3506   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3507   \@@_actually_draw_Ldots:
3508   \group_end:
3509 }
3510 }

```

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`.

```

3511 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3512 {
3513   \bool_if:NTF \l_@@_initial_open_bool
3514   {
3515     \@@_open_x_initial_dim:
3516     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3517     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3518   }
3519   { \@@_set_initial_coords_from_anchor:n { base~east } }
3520   \bool_if:NTF \l_@@_final_open_bool
3521   {
3522     \@@_open_x_final_dim:
3523     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3524     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3525   }
3526   { \@@_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 (?).

```

3527   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3528   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3529   \@@_draw_line:
3530 }

```

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.

```

3531 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3532 {
3533   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3534   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3535   {
3536     \@@_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.

```

3537   \group_begin:
3538   \int_compare:nNnTF { #1 } = 0
3539   { \color { nicematrix-first-row } }
3540   {

```

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.

```

3541     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3542     { \color { nicematrix-last-row } }
3543   }
3544   \keys_set:nn { NiceMatrix / xdots } { #3 }
3545   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3546   \@@_actually_draw_Cdots:
3547   \group_end:
3548 }
3549 }

```

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.

```

3550 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3551 {
3552   \bool_if:NTF \l_@@_initial_open_bool
3553     { \@@_open_x_initial_dim: }
3554     { \@@_set_initial_coords_from_anchor:n { mid-east } }
3555   \bool_if:NTF \l_@@_final_open_bool
3556     { \@@_open_x_final_dim: }
3557     { \@@_set_final_coords_from_anchor:n { mid-west } }
3558   \bool_lazy_and:nnTF
3559     \l_@@_initial_open_bool
3560     \l_@@_final_open_bool
3561   {
3562     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3563     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3564     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
3565     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3566     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3567   }
3568   {
3569     \bool_if:NT \l_@@_initial_open_bool
3570     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3571     \bool_if:NT \l_@@_final_open_bool
3572     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3573   }
3574   \@@_draw_line:
3575 }

3576 \cs_new_protected:Npn \@@_open_y_initial_dim:
3577 {
3578   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3579   \dim_set:Nn \l_@@_y_initial_dim
3580     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3581   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3582     {
3583       \cs_if_exist:cT
3584         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3585         {
3586           \pgfpointanchor
3587             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3588             { north }
3589           \dim_set:Nn \l_@@_y_initial_dim
3590             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3591         }
3592     }
3593 }

3594 \cs_new_protected:Npn \@@_open_y_final_dim:
3595 {
3596   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3597   \dim_set:Nn \l_@@_y_final_dim
3598     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3599   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3600     {
3601       \cs_if_exist:cT
3602         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3603         {
3604           \pgfpointanchor
3605             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }

```



```

3606         { south }
3607         \dim_set:Nn \l_@@_y_final_dim
3608         { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3609     }
3610 }
3611 }

```

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.

```

3612 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3613 {
3614     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3615     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3616     {
3617         \@@_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.

```

3618     \group_begin:
3619     \int_compare:nNnTF { #2 } = 0
3620     { \color { nicematrix-first-col } }
3621     {
3622         \int_compare:nNnT { #2 } = \l_@@_last_col_int
3623         { \color { nicematrix-last-col } }
3624     }
3625     \keys_set:nn { NiceMatrix / xdots } { #3 }
3626     \tl_if_empty:VF \l_@@_xdots_color_tl
3627     { \color { \l_@@_xdots_color_tl } }
3628     \@@_actually_draw_Vdots:
3629 \group_end:
3630 }
3631 }

```

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`.

```

3632 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3633 {

```

The boolean `\l_tmpa_bool` indicates whether the column is of type 1 or may be considered as if.

```

3634     \bool_set_false:N \l_tmpa_bool

```

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

```

3635     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3636     {
3637         \@@_set_initial_coords_from_anchor:n { south-west }
3638         \@@_set_final_coords_from_anchor:n { north-west }
3639         \bool_set:Nn \l_tmpa_bool
3640         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3641     }

```

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

```

3642 \bool_if:NTF \l_@@_initial_open_bool
3643 \@@_open_y_initial_dim:
3644 { \@@_set_initial_coords_from_anchor:n { south } }
3645 \bool_if:NTF \l_@@_final_open_bool
3646 \@@_open_y_final_dim:
3647 { \@@_set_final_coords_from_anchor:n { north } }
3648 \bool_if:NTF \l_@@_initial_open_bool
3649 {
3650 \bool_if:NTF \l_@@_final_open_bool
3651 {
3652 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3653 \dim_set_eq:NN \l_tmpa_dim \pgf@x
3654 \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
3655 \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3656 \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.

```

3657 \int_compare:nNnT \l_@@_last_col_int > { -2 }
3658 {
3659 \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3660 {
3661 \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3662 \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3663 \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3664 \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3665 }
3666 }
3667 }
3668 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3669 }
3670 {
3671 \bool_if:NTF \l_@@_final_open_bool
3672 { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3673 }

```

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.

```

3674 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3675 {
3676 \dim_set:Nn \l_@@_x_initial_dim
3677 {
3678 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3679 \l_@@_x_initial_dim \l_@@_x_final_dim
3680 }
3681 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3682 }
3683 }
3684 }
3685 \@@_draw_line:
3686 }

```

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.

```

3687 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3688 {
3689 \@@_adjust_to_submatrix:nn { #1 } { #2 }

```

```

3690 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3691 {
3692     \@@_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.

```

3693     \group_begin:
3694     \keys_set:nn { NiceMatrix / xdots } { #3 }
3695     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3696     \@@_actually_draw_Ddots:
3697 \group_end:
3698 }
3699 }

```

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`.

```

3700 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3701 {
3702     \bool_if:NTF \l_@@_initial_open_bool
3703     {
3704         \@@_open_y_initial_dim:
3705         \@@_open_x_initial_dim:
3706     }
3707     { \@@_set_initial_coords_from_anchor:n { south~east } }
3708     \bool_if:NTF \l_@@_final_open_bool
3709     {
3710         \@@_open_x_final_dim:
3711         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3712     }
3713     { \@@_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.

```

3714     \bool_if:NT \l_@@_parallelize_diags_bool
3715     {
3716         \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).

```

3717         \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.

```

3718     {
3719         \dim_gset:Nn \g_@@_delta_x_one_dim
3720         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3721         \dim_gset:Nn \g_@@_delta_y_one_dim
3722         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3723     }

```

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`.

```

3724     {
3725         \dim_set:Nn \l_@@_y_final_dim
3726         {
3727             \l_@@_y_initial_dim +
3728             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3729             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3730         }
3731     }
3732 }
3733 \@@_draw_line:
3734 }
```

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.

```

3735 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3736 {
3737     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3738     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3739     {
3740         \@@_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.

```

3741     \group_begin:
3742     \keys_set:nn { NiceMatrix / xdots } { #3 }
3743     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3744     \@@_actually_draw_Iddots:
3745     \group_end:
3746 }
3747 }
```

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`.

```

3748 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3749 {
3750     \bool_if:NTF \l_@@_initial_open_bool
3751     {
3752         \@@_open_y_initial_dim:
3753         \@@_open_x_initial_dim:
3754     }
3755     { \@@_set_initial_coords_from_anchor:n { south-west } }
3756     \bool_if:NTF \l_@@_final_open_bool
3757     {
3758         \@@_open_y_final_dim:
3759         \@@_open_x_final_dim:
3760     }
3761     { \@@_set_final_coords_from_anchor:n { north-east } }
3762     \bool_if:NT \l_@@_parallelize_diags_bool
3763     {
```

```

3764 \int_gincr:N \g_@@_iddots_int
3765 \int_compare:nNnTF \g_@@_iddots_int = 1
3766 {
3767     \dim_gset:Nn \g_@@_delta_x_two_dim
3768     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3769     \dim_gset:Nn \g_@@_delta_y_two_dim
3770     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3771 }
3772 {
3773     \dim_set:Nn \l_@@_y_final_dim
3774     {
3775         \l_@@_y_initial_dim +
3776         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3777         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3778     }
3779 }
3780 }
3781 \@@_draw_line:
3782 }

```

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`

```

3783 \cs_new_protected:Npn \@@_draw_line:
3784 {
3785     \pgfrememberpicturerepositiononpagetrue
3786     \pgf@relevantforpicturesizefalse
3787     \bool_lazy_or:nnTF
3788     { \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.

```

3789     \l_@@_dotted_bool
3790     \@@_draw_standard_dotted_line:
3791     \@@_draw_unstandard_dotted_line:
3792 }

```

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.

```

3793 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
3794 {
3795     \begin { scope }
3796     \exp_args:No \@@_draw_unstandard_dotted_line:n
3797     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3798 }

```

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.

```

3799 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
3800 {
3801   \@@_draw_unstandard_dotted_line:nVV
3802   { #1 }
3803   \l_@@_xdots_up_tl
3804   \l_@@_xdots_down_tl
3805 }

3806 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
3807 {
3808   \draw
3809   [
3810     #1 ,
3811     shorten~> = \l_@@_xdots_shorten_dim ,
3812     shorten~< = \l_@@_xdots_shorten_dim ,
3813   ]
3814   ( \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).

```

3815   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3816   node [ sloped , below ] { $ \scriptstyle #3 $ }
3817   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3818   \end { scope }
3819 }
3820 \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).

```

3821 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3822 {
3823   \bool_lazy_and:nnF
3824   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3825   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3826   {
3827     \pgfscope
3828     \pgftransformshift
3829     {
3830       \pgfpointlineattime { 0.5 }
3831       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3832       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3833     }
3834     \pgftransformrotate
3835     {
3836       \fp_eval:n
3837       {
3838         atand
3839         (
3840           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3841           \l_@@_x_final_dim - \l_@@_x_initial_dim
3842         )
3843       }
3844     }
3845     \pgfnode
3846     { rectangle }
3847     { south }
3848     {
3849       \c_math_toggle_token
3850       \scriptstyle \l_@@_xdots_up_tl
3851       \c_math_toggle_token

```

```

3852     }
3853     { }
3854     { \pgfusepath { } }
3855     \pgfnode
3856     { rectangle }
3857     { north }
3858     {
3859         \c_math_toggle_token
3860         \scriptstyle \l_@@_xdots_down_tl
3861         \c_math_toggle_token
3862     }
3863     { }
3864     { \pgfusepath { } }
3865     \endpgfscope
3866 }
3867 \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.

```

3868     \dim_zero_new:N \l_@@_l_dim
3869     \dim_set:Nn \l_@@_l_dim
3870     {
3871         \fp_to_dim:n
3872         {
3873             sqrt
3874             (
3875                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3876                 +
3877                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3878             )
3879         }
3880     }

```

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.

```

3881     \bool_lazy_or:nnF
3882     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3883     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3884     \@@_draw_standard_dotted_line_i:
3885 \group_end:
3886 }
3887 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3888 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3889 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

3890     \bool_if:NTF \l_@@_initial_open_bool
3891     {
3892         \bool_if:NTF \l_@@_final_open_bool
3893         {
3894             \int_set:Nn \l_tmpa_int
3895             { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3896         }
3897         {
3898             \int_set:Nn \l_tmpa_int
3899             {
3900                 \dim_ratio:nn
3901                 { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3902                 \l_@@_inter_dots_dim
3903             }
3904         }

```

```

3905     }
3906     {
3907         \bool_if:NTF \l_@@_final_open_bool
3908         {
3909             \int_set:Nn \l_tmpa_int
3910             {
3911                 \dim_ratio:nn
3912                 { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3913                 \l_@@_inter_dots_dim
3914             }
3915         }
3916         {
3917             \int_set:Nn \l_tmpa_int
3918             {
3919                 \dim_ratio:nn
3920                 { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3921                 \l_@@_inter_dots_dim
3922             }
3923         }
3924     }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

3925     \dim_set:Nn \l_tmpa_dim
3926     {
3927         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3928         \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3929     }
3930     \dim_set:Nn \l_tmpb_dim
3931     {
3932         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3933         \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3934     }

```

The length ℓ is the length of the dotted line. We note Δ the length between two dots and n the number of intervals between dots. We note $\delta = \frac{1}{2}(\ell - n\Delta)$. The distance between the initial extremity of the line and the first dot will be equal to $k \cdot \delta$ where $k = 0, 1$ or 2 . We first compute this number k in `\l_tmpb_int`.

```

3935     \int_set:Nn \l_tmpb_int
3936     {
3937         \bool_if:NTF \l_@@_initial_open_bool
3938         { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3939         { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3940     }

```

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.

```

3941     \dim_gadd:Nn \l_@@_x_initial_dim
3942     {
3943         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3944         \dim_ratio:nn
3945         { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3946         { 2 \l_@@_l_dim }
3947         * \l_tmpb_int
3948     }
3949     \dim_gadd:Nn \l_@@_y_initial_dim
3950     {
3951         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3952         \dim_ratio:nn
3953         { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3954         { 2 \l_@@_l_dim }
3955         * \l_tmpb_int
3956     }
3957     \pgf@relevantforpicturesizefalse

```



```

3958 \int_step_inline:nnn 0 \l_tmpa_int
3959 {
3960   \pgfpathcircle
3961   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3962   { \l_@@_radius_dim }
3963   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3964   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3965 }
3966 \pgfusepathqfill
3967 }

```

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.

```

3968 \hook_gput_code:nnn { begindocument } { . }
3969 {
3970   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3971   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3972   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3973   {
3974     \int_compare:nNnTF \c@jCol = 0
3975     { \@@_error:nn { in~first~col } \Ldots }
3976     {
3977       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3978       { \@@_error:nn { in~last~col } \Ldots }
3979       {
3980         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
3981         { #1 , down = #2 , up = #3 }
3982       }
3983     }
3984     \bool_if:NF \l_@@_nullify_dots_bool
3985     { \phantom { \ensuremath { \@@_old_ldots } } }
3986     \bool_gset_true:N \g_@@_empty_cell_bool
3987   }

3988   \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3989   {
3990     \int_compare:nNnTF \c@jCol = 0
3991     { \@@_error:nn { in~first~col } \Cdots }
3992     {
3993       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3994       { \@@_error:nn { in~last~col } \Cdots }
3995       {
3996         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
3997         { #1 , down = #2 , up = #3 }
3998       }
3999     }
4000     \bool_if:NF \l_@@_nullify_dots_bool
4001     { \phantom { \ensuremath { \@@_old_cdots } } }
4002     \bool_gset_true:N \g_@@_empty_cell_bool
4003   }

```

```

4004 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4005 {
4006   \int_compare:nNnTF \c@iRow = 0
4007   { \@@_error:nn { in~first~row } \Vdots }
4008   {
4009     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4010     { \@@_error:nn { in~last~row } \Vdots }
4011     {
4012       \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4013       { #1 , down = #2 , up = #3 }
4014     }
4015   }
4016   \bool_if:NF \l_@@_nullify_dots_bool
4017   { \phantom { \ensuremath { \@@_old_vdots } } }
4018   \bool_gset_true:N \g_@@_empty_cell_bool
4019 }

4020 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4021 {
4022   \int_case:nnF \c@iRow
4023   {
4024     0 { \@@_error:nn { in~first~row } \Ddots }
4025     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4026   }
4027   {
4028     \int_case:nnF \c@jCol
4029     {
4030       0 { \@@_error:nn { in~first~col } \Ddots }
4031       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4032     }
4033     {
4034       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4035       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
4036       { #1 , down = #2 , up = #3 }
4037     }
4038   }
4039 }
4040 \bool_if:NF \l_@@_nullify_dots_bool
4041 { \phantom { \ensuremath { \@@_old_ddots } } }
4042 \bool_gset_true:N \g_@@_empty_cell_bool
4043 }

4044 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4045 {
4046   \int_case:nnF \c@iRow
4047   {
4048     0 { \@@_error:nn { in~first~row } \Iddots }
4049     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4050   }
4051   {
4052     \int_case:nnF \c@jCol
4053     {
4054       0 { \@@_error:nn { in~first~col } \Iddots }
4055       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4056     }
4057     {
4058       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4059       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Iddots }
4060       { #1 , down = #2 , up = #3 }
4061     }
4062   }
4063   \bool_if:NF \l_@@_nullify_dots_bool

```

```

4064         { \phantom { \ensuremath { \@@_old_iddots } } }
4065         \bool_gset_true:N \g_@@_empty_cell_bool
4066     }
4067 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

4068 \keys_define:nn { NiceMatrix / Ddots }
4069 {
4070     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4071     draw-first .default:n = true ,
4072     draw-first .value_forbidden:n = true
4073 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

4074 \cs_new_protected:Npn \@@_Hspace:
4075 {
4076     \bool_gset_true:N \g_@@_empty_cell_bool
4077     \hspace
4078 }

```

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.

```

4079 \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.

```

4080 \cs_new:Npn \@@_Hdotsfor:
4081 {
4082     \bool_lazy_and:nnTF
4083     { \int_compare_p:nNn \c@jCol = 0 }
4084     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4085     {
4086         \bool_if:NTF \g_@@_after_col_zero_bool
4087         {
4088             \multicolumn { 1 } { c } { }
4089             \@@_Hdotsfor_i
4090         }
4091         { \@@_fatal:n { Hdotsfor~in~col~0 } }
4092     }
4093     {
4094         \multicolumn { 1 } { c } { }
4095         \@@_Hdotsfor_i
4096     }
4097 }

```

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:).

```

4098 \hook_gput_code:nnn { begindocument } { . }
4099 {
4100     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4101     \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.

```

4102     \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4103     {
4104         \tl_gput_right:Nx \g_@@_Hvdotsfor_lines_tl

```

```

4105     {
4106         \@@_Hdotsfor:nnnn
4107         { \int_use:N \c@iRow }
4108         { \int_use:N \c@jCol }
4109         { #2 }
4110         {
4111             #1 , #3 ,
4112             down = \exp_not:n { #4 } ,
4113             up = \exp_not:n { #5 }
4114         }
4115     }
4116     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4117 }
4118 }

```

Enf of \AddToHook.

```

4119 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4120 {
4121     \bool_set_false:N \l_@@_initial_open_bool
4122     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4123     \int_set:Nn \l_@@_initial_i_int { #1 }
4124     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

4125     \int_compare:nNnTF { #2 } = 1
4126     {
4127         \int_set:Nn \l_@@_initial_j_int 1
4128         \bool_set_true:N \l_@@_initial_open_bool
4129     }
4130     {
4131         \cs_if_exist:cTF
4132         {
4133             pgf @ sh @ ns @ \@@_env:
4134             - \int_use:N \l_@@_initial_i_int
4135             - \int_eval:n { #2 - 1 }
4136         }
4137         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4138         {
4139             \int_set:Nn \l_@@_initial_j_int { #2 }
4140             \bool_set_true:N \l_@@_initial_open_bool
4141         }
4142     }
4143     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4144     {
4145         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4146         \bool_set_true:N \l_@@_final_open_bool
4147     }
4148     {
4149         \cs_if_exist:cTF
4150         {
4151             pgf @ sh @ ns @ \@@_env:
4152             - \int_use:N \l_@@_final_i_int
4153             - \int_eval:n { #2 + #3 }
4154         }
4155         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4156         {
4157             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4158             \bool_set_true:N \l_@@_final_open_bool
4159         }
4160     }
4161     \group_begin:

```

```

4162 \int_compare:nNnTF { #1 } = 0
4163   { \color { nicematrix-first-row } }
4164   {
4165     \int_compare:nNnT { #1 } = \g_@@_row_total_int
4166     { \color { nicematrix-last-row } }
4167   }
4168 \keys_set:nn { NiceMatrix / xdots } { #4 }
4169 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4170 \@@_actually_draw_Ldots:
4171 \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).

```

4172 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4173   { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4174 }

4175 \hook_gput_code:nnn { begindocument } { . }
4176 {
4177   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4178   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4179   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4180   {
4181     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4182     {
4183       \@@_Vdotsfor:nnnn
4184       { \int_use:N \c@iRow }
4185       { \int_use:N \c@jCol }
4186       { #2 }
4187       {
4188         #1 , #3 ,
4189         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4190       }
4191     }
4192   }
4193 }

```

Enf of `\AddToHook`.

```

4194 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4195 {
4196   \bool_set_false:N \l_@@_initial_open_bool
4197   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

4198 \int_set:Nn \l_@@_initial_j_int { #2 }
4199 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it’s a bit more complicated.

```

4200 \int_compare:nNnTF #1 = 1
4201   {
4202     \int_set:Nn \l_@@_initial_i_int 1
4203     \bool_set_true:N \l_@@_initial_open_bool
4204   }
4205   {
4206     \cs_if_exist:cTF
4207     {
4208       pgf @ sh @ ns @ \@@_env:
4209       - \int_eval:n { #1 - 1 }
4210       - \int_use:N \l_@@_initial_j_int
4211     }
4212     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4213   }

```

```

4214         \int_set:Nn \l_@@_initial_i_int { #1 }
4215         \bool_set_true:N \l_@@_initial_open_bool
4216     }
4217 }
4218 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4219 {
4220     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4221     \bool_set_true:N \l_@@_final_open_bool
4222 }
4223 {
4224     \cs_if_exist:cTF
4225     {
4226         pgf @ sh @ ns @ \@@_env:
4227         - \int_eval:n { #1 + #3 }
4228         - \int_use:N \l_@@_final_j_int
4229     }
4230     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4231     {
4232         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4233         \bool_set_true:N \l_@@_final_open_bool
4234     }
4235 }
4236 \group_begin:
4237 \int_compare:nNnTF { #2 } = 0
4238 { \color { nicematrix-first-col } }
4239 {
4240     \int_compare:nNnT { #2 } = \g_@@_col_total_int
4241     { \color { nicematrix-last-col } }
4242 }
4243 \keys_set:nn { NiceMatrix / xdots } { #4 }
4244 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4245 \@@_actually_draw_Vdots:
4246 \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).

```

4247     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4248     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4249 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

4250 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

```

The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells.

First, we write a command with an argument of the format i - j and applies the command `\int_eval:n` to i and j ; this must *not* be protected (and is, of course fully expandable).⁷⁰

```

4251 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4252 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

⁷⁰Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

4253 \hook_gput_code:nnn { begindocument } { . }
4254 {
4255   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4256   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4257   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4258     {
4259       \group_begin:
4260       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4261       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4262       \use:e
4263       {
4264         \@@_line_i:nn
4265         { \@@_double_int_eval:n #2 \q_stop }
4266         { \@@_double_int_eval:n #3 \q_stop }
4267       }
4268       \group_end:
4269     }
4270 }
4271 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4272 {
4273   \bool_set_false:N \l_@@_initial_open_bool
4274   \bool_set_false:N \l_@@_final_open_bool
4275   \bool_if:nTF
4276     {
4277       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4278       ||
4279       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4280     }
4281     {
4282       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4283     }
4284     { \@@_draw_line_ii:nn { #1 } { #2 } }
4285 }
4286 \hook_gput_code:nnn { begindocument } { . }
4287 {
4288   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4289   {

```

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:`.

```

4290   \c_@@_pgfortikzpicture_tl
4291   \@@_draw_line_iii:nn { #1 } { #2 }
4292   \c_@@_endpgfortikzpicture_tl
4293 }
4294 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

4295 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4296 {
4297   \pgfrememberpicturepositiononpagetrue
4298   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4299   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4300   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4301   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4302   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4303   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4304   \@@_draw_line:
4305 }

```

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`

```

4306 \keys_define:nn { NiceMatrix / RowStyle }
4307 {
4308   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4309   cell-space-top-limit .initial:n = \c_zero_dim ,
4310   cell-space-top-limit .value_required:n = true ,
4311   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4312   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4313   cell-space-bottom-limit .value_required:n = true ,
4314   cell-space-limits .meta:n =
4315   {
4316     cell-space-top-limit = #1 ,
4317     cell-space-bottom-limit = #1 ,
4318   } ,
4319   color .tl_set:N = \l_tmpa_tl ,
4320   color .value_required:n = true ,
4321   bold .bool_set:N = \l_tmpa_bool ,
4322   bold .default:n = true ,
4323   bold .initial:n = false ,
4324   nb-rows .int_set:N = \l_@@_key_nb_rows_int ,
4325   nb-rows .value_required:n = true ,
4326   nb-rows .initial:n = 1 ,
4327   rowcolor .tl_set:N = \l_@@_tmpc_tl ,
4328   rowcolor .value_required:n = true ,
4329   rowcolor .initial:n = ,
4330   unknown .code:n = \@@_error:n { Unknown~key-for-RowStyle }
4331 }

```

```

4332 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4333 {
4334   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

4335   \tl_if_empty:NF \l_@@_tmpc_tl
4336   {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

4337     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4338     {
4339       \@@_rectanglecolor
4340       { \l_@@_tmpc_tl }
4341       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4342       { \int_use:N \c@iRow - * }
4343     }

```

Then, the other rows (if there is several rows).

```

4344     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4345     {
4346       \tl_gput_right:Nx \g_nicematrix_code_before_tl
4347       {
4348         \@@_rowcolor
4349         { \l_@@_tmpc_tl }
4350         {
4351           \int_eval:n { \c@iRow + 1 }
4352           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4353         }
4354       }
4355     }

```



```

4356     }
4357     \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4358     \tl_gput_right:Nx \g_@@_row_style_tl
4359     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4360     \tl_gput_right:Nn \g_@@_row_style_tl { #2 }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
4361     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4362     {
4363         \tl_gput_right:Nx \g_@@_row_style_tl
4364         {
4365             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4366             {
4367                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
4368                 { \dim_use:N \l_tmpa_dim }
4369             }
4370         }
4371     }
\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.
4372     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4373     {
4374         \tl_gput_right:Nx \g_@@_row_style_tl
4375         {
4376             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4377             {
4378                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4379                 { \dim_use:N \l_tmpb_dim }
4380             }
4381         }
4382     }
\l_tmpa_tl is the value of the key color of \RowStyle.
4383     \tl_if_empty:NF \l_tmpa_tl
4384     {
4385         \tl_gput_right:Nx \g_@@_row_style_tl
4386         { \mode_leave_vertical: \exp_not:N \color { \l_tmpa_tl } }
4387     }
\l_tmpa_bool is the value of the key bold.
4388     \bool_if:NT \l_tmpa_bool
4389     {
4390         \tl_gput_right:Nn \g_@@_row_style_tl
4391         {
4392             \if_mode_math:
4393             \c_math_toggle_token
4394             \bfseries \boldmath
4395             \c_math_toggle_token
4396             \else:
4397             \bfseries \boldmath
4398             \fi:
4399         }
4400     }
4401     \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4402     \g_@@_row_style_tl
4403     \ignorespaces
4404 }

```

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).

```
4405 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4406 {
```

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.

```
4407 \int_zero:N \l_tmpa_int
4408 \seq_map_indexed_inline:Nn \g_@@_colors_seq
4409 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4410 \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

First, the case where the color is a *new* color (not in the sequence).

```
4411 {
4412 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4413 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4414 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
4415 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4416 }

4417 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4418 \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`).

```
4419 \cs_new_protected:Npn \@@_actually_color:
4420 {
4421 \pgfpicture
4422 \pgf@relevantforpicturesizefalse
4423 \seq_map_indexed_inline:Nn \g_@@_colors_seq
4424 {
4425 \color ##2
4426 \use:c { g_@@_color _ ##1 _tl }
4427 \tl_gclear:c { g_@@_color _ ##1 _tl }
4428 \pgfusepath { fill }
4429 }
4430 \endpgfpicture
4431 }

4432 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4433 {
4434 \tl_set:Nn \l_@@_rows_tl { #1 }
4435 \tl_set:Nn \l_@@_cols_tl { #2 }
4436 \@@_cartesian_path:
4437 }
```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
4438 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4439 {
4440   \tl_if_blank:nF { #2 }
4441   {
4442     \@@_add_to_colors_seq:xn
4443     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4444     { \@@_cartesian_color:nn { #3 } { - } }
4445   }
4446 }
```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```
4447 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4448 {
4449   \tl_if_blank:nF { #2 }
4450   {
4451     \@@_add_to_colors_seq:xn
4452     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4453     { \@@_cartesian_color:nn { - } { #3 } }
4454   }
4455 }
```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
4456 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4457 {
4458   \tl_if_blank:nF { #2 }
4459   {
4460     \@@_add_to_colors_seq:xn
4461     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4462     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4463   }
4464 }
```

The last argument is the radius of the corners of the rectangle.

```
4465 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4466 {
4467   \tl_if_blank:nF { #2 }
4468   {
4469     \@@_add_to_colors_seq:xn
4470     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4471     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4472   }
4473 }
```

The last argument is the radius of the corners of the rectangle.

```
4474 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4475 {
4476   \@@_cut_on_hyphen:w #1 \q_stop
4477   \tl_clear_new:N \l_@@_tmpc_tl
4478   \tl_clear_new:N \l_@@_tmpd_tl
4479   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
4480   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
4481   \@@_cut_on_hyphen:w #2 \q_stop
4482   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
4483   \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`.

```
4484   \@@_cartesian_path:n { #3 }
4485 }
```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

4486 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4487 {
4488   \clist_map_inline:nn { #3 }
4489     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4490 }

4491 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4492 {
4493   \int_step_inline:nn { \int_use:N \c@iRow }
4494     {
4495       \int_step_inline:nn { \int_use:N \c@jCol }
4496         {
4497           \int_if_even:nTF { ####1 + ##1 }
4498             { \@@_cellcolor [ #1 ] { #2 } }
4499             { \@@_cellcolor [ #1 ] { #3 } }
4500           { ##1 - ####1 }
4501         }
4502     }
4503 }

```

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”.

```

4504 \NewDocumentCommand \@@_arraycolor { 0 { } m }
4505 {
4506   \@@_rectanglecolor [ #1 ] { #2 }
4507   { 1 - 1 }
4508   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4509 }

4510 \keys_define:nn { NiceMatrix / rowcolors }
4511 {
4512   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4513   respect-blocks .default:n = true ,
4514   cols .tl_set:N = \l_@@_cols_tl ,
4515   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4516   restart .default:n = true ,
4517   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4518 }

```

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*.

```

4519 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
4520 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

4521   \group_begin:
4522   \seq_clear_new:N \l_@@_colors_seq
4523   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
4524   \tl_clear_new:N \l_@@_cols_tl
4525   \tl_set:Nn \l_@@_cols_tl { - }
4526   \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).

```

4527 \int_zero_new:N \l_@@_color_int
4528 \int_set:Nn \l_@@_color_int 1
4529 \bool_if:NT \l_@@_respect_blocks_bool
4530 {

```

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`).

```

4531 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4532 \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
4533 { \@@_not_in_exterior_p:nnnnn ##1 }
4534 }
4535 \pgfpicture
4536 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

4537 \clist_map_inline:nn { #2 }
4538 {
4539 \tl_set:Nn \l_tmpa_tl { ##1 }
4540 \tl_if_in:NnTF \l_tmpa_tl { - }
4541 { \@@_cut_on_hyphen:w ##1 \q_stop }
4542 { \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.

```

4543 \int_set:Nn \l_tmpa_int \l_tmpa_tl
4544 \bool_if:NTF \l_@@_rowcolors_restart_bool
4545 { \int_set:Nn \l_@@_color_int 1 }
4546 { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
4547 \int_zero_new:N \l_@@_tmpc_int
4548 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
4549 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
4550 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4551 \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).

```

4552 \bool_if:NT \l_@@_respect_blocks_bool
4553 {
4554 \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
4555 { \@@_intersect_our_row_p:nnnnn ####1 }
4556 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4557 }
4558 \tl_set:Nx \l_@@_rows_tl
4559 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

4560 \tl_clear_new:N \l_@@_color_tl
4561 \tl_set:Nx \l_@@_color_tl
4562 {
4563 \@@_color_index:n
4564 {
4565 \int_mod:nn
4566 { \l_@@_color_int - 1 }
4567 { \seq_count:N \l_@@_colors_seq }
4568 + 1
4569 }
4570 }
4571 \tl_if_empty:NF \l_@@_color_tl
4572 {
4573 \@@_add_to_colors_seq:xx

```

```

4574         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
4575         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
4576     }
4577     \int_incr:N \l_@@_color_int
4578     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4579 }
4580 }
4581 \endpgfpicture
4582 \group_end:
4583 }

```

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.

```

4584 \cs_new:Npn \@@_color_index:n #1
4585 {
4586     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
4587     { \@@_color_index:n { #1 - 1 } }
4588     { \seq_item:Nn \l_@@_colors_seq { #1 } }
4589 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

4590 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4591 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

```

```

4592 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
4593 {
4594     \int_compare:nNnT { #3 } > \l_tmpb_int
4595     { \int_set:Nn \l_tmpb_int { #3 } }
4596 }

```

```

4597 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
4598 {
4599     \bool_lazy_or:nnTF
4600     { \int_compare_p:nNn { #4 } = \c_zero_int }
4601     { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
4602     \prg_return_false:
4603     \prg_return_true:
4604 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

4605 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
4606 {
4607     \bool_if:nTF
4608     {
4609         \int_compare_p:n { #1 <= \l_tmpa_int }
4610         &&
4611         \int_compare_p:n { \l_tmpa_int <= #3 }
4612     }
4613     \prg_return_true:
4614     \prg_return_false:
4615 }

```

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`).

```

4616 \cs_new_protected:Npn \@@_cartesian_path:n #1
4617 {
4618   \bool_lazy_and:nnT
4619   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4620   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4621   {
4622     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4623     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4624   }

```

We begin the loop over the columns.

```

4625   \clist_map_inline:Nn \l_@@_cols_tl
4626   {
4627     \tl_set:Nn \l_tmpa_tl { ##1 }
4628     \tl_if_in:NnTF \l_tmpa_tl { - }
4629     { \@@_cut_on_hyphen:w ##1 \q_stop }
4630     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4631     \bool_lazy_or:nnT
4632     { \tl_if_blank_p:V \l_tmpa_tl }
4633     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4634     { \tl_set:Nn \l_tmpa_tl { 1 } }
4635     \bool_lazy_or:nnT
4636     { \tl_if_blank_p:V \l_tmpb_tl }
4637     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4638     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4639     \int_compare:nNnT \l_tmpb_tl > \c@jCol
4640     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

\l_@@_tmpc_tl will contain the number of column.

```

4641     \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.

```

4642     \@@_qpoint:n { col - \l_tmpa_tl }
4643     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4644     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4645     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4646     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
4647     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4648   \clist_map_inline:Nn \l_@@_rows_tl
4649   {
4650     \tl_set:Nn \l_tmpa_tl { #####1 }
4651     \tl_if_in:NnTF \l_tmpa_tl { - }
4652     { \@@_cut_on_hyphen:w #####1 \q_stop }
4653     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4654     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4655     \tl_if_empty:NT \l_tmpb_tl
4656     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4657     \int_compare:nNnT \l_tmpb_tl > \c@iRow
4658     { \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.

```

4659     \seq_if_in:NxF \l_@@_corners_cells_seq
4660     { \l_tmpa_tl - \l_@@_tmpc_tl }
4661     {
4662       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
4663       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4664       \@@_qpoint:n { row - \l_tmpa_tl }
4665       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4666       \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4667       \pgfpathrectanglecorners
4668       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }

```

```

4669         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4670     }
4671 }
4672 }
4673 }

```

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`).

```

4674 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to 2,4-6,8-* and `\c@jCol` equal to 10, the clist `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

4675 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4676 {
4677   \clist_set_eq:NN \l_tmpa_clist #1
4678   \clist_clear:N #1
4679   \clist_map_inline:Nn \l_tmpa_clist
4680   {
4681     \tl_set:Nn \l_tmpa_tl { ##1 }
4682     \tl_if_in:NnTF \l_tmpa_tl { - }
4683     { \@@_cut_on_hyphen:w ##1 \q_stop }
4684     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4685     \bool_lazy_or:nnT
4686     { \tl_if_blank_p:V \l_tmpa_tl }
4687     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4688     { \tl_set:Nn \l_tmpa_tl { 1 } }
4689     \bool_lazy_or:nnT
4690     { \tl_if_blank_p:V \l_tmpb_tl }
4691     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4692     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4693     \int_compare:nNnT \l_tmpb_tl > #2
4694     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4695     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4696     { \clist_put_right:Nn #1 { #####1 } }
4697   }
4698 }

```

When the user uses the key `colortbl-like`, the following command will be linked to `\cellcolor` in the tabular.

```

4699 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4700 {
4701   \peek_remove_spaces:n
4702   {
4703     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4704     {

```

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`).

```

4705       \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
4706       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4707     }
4708   }
4709 }

```

When the user uses the key `colortbl-like`, the following command will be linked to `\rowcolor` in the tabular.

```

4710 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4711 {
4712   \peek_remove_spaces:n
4713   {

```



```

4714 \tl_gput_right:Nx \g_nicematrix_code_before_tl
4715 {
4716   \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4717   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4718   { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4719 }
4720 }
4721 }

```

```

4722 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4723 {

```

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).

```

4724 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4725 {

```

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).

```

4726 \tl_gput_left:Nx \g_nicematrix_code_before_tl
4727 {
4728   \exp_not:N \columncolor [ #1 ]
4729   { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4730 }
4731 }
4732 }

```

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`.

```

4733 \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`.

```

4734 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4735 {
4736   \int_compare:nNnTF \l_@@_first_col_int = 0
4737   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4738   {
4739     \int_compare:nNnTF \c@jCol = 0
4740     {
4741       \int_compare:nNnF \c@iRow = { -1 }
4742       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4743     }
4744     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4745   }
4746 }

```

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.

```

4747 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4748 {
4749   \int_compare:nNnF \c@iRow = 0
4750     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4751 }

```

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.

```

4752 \keys_define:nn { NiceMatrix / Rules }
4753 {
4754   position .int_set:N = \l_@@_position_int ,
4755   position .value_required:n = true ,
4756   start .int_set:N = \l_@@_start_int ,
4757   start .initial:n = 1 ,
4758   end .int_set:N = \l_@@_end_int ,

```

The following keys are no-op because there are keys which may be inherited from a list of pairs *key=value* of a definition of a customized rule (with the key `custom-line` of `\NiceMatrixOptions`).

```

4759   letter .code:n = \prg_do_nothing: ,
4760   command .code:n = \prg_do_nothing:
4761 }

```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

4762 \keys_define:nn { NiceMatrix / RulesBis }
4763 {
4764   multiplicity .int_set:N = \l_@@_multiplicity_int ,
4765   multiplicity .initial:n = 1 ,
4766   dotted .bool_set:N = \l_@@_dotted_bool ,
4767   dotted .initial:n = false ,
4768   dotted .default:n = true ,
4769   color .code:n = \@@_set_CT@arc@: #1 \q_stop ,
4770   color .value_required:n = true ,
4771   sep-color .code:n = \@@_set_CT@drsc@: #1 \q_stop ,
4772   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.

```

4773   tikz .tl_set:N = \l_@@_tikz_rule_tl ,
4774   tikz .value_required:n = true ,
4775   tikz .initial:n = ,
4776   width .dim_set:N = \l_@@_rule_width_dim ,
4777   width .value_required:n = true
4778 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```
4779 \cs_new_protected:Npn \@@_vline:n #1
4780 {
```

The group is for the options.

```
4781   \group_begin:
4782   \int_zero_new:N \l_@@_end_int
4783   \int_set_eq:NN \l_@@_end_int \c@iRow
4784   \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).

```
4785   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
4786   \@@_vline_i:
4787   \group_end:
4788 }
```

```
4789 \cs_new_protected:Npn \@@_vline_i:
4790 {
4791   \int_zero_new:N \l_@@_local_start_int
4792   \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`.

```
4793   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
4794   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
4795   \l_tmpa_tl
4796   {
```

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.

```
4797   \bool_gset_true:N \g_tmpa_bool
4798   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4799   { \@@_test_vline_in_block:nnnnn ##1 }
4800   \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4801   { \@@_test_vline_in_block:nnnnn ##1 }
4802   \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4803   { \@@_test_vline_in_stroken_block:nnnn ##1 }
4804   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
4805   \bool_if:NTF \g_tmpa_bool
4806   {
4807     \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.

```
4808     { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
4809   }
4810   {
4811     \int_compare:nNnT \l_@@_local_start_int > 0
4812     {
4813       \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
4814       \@@_vline_ii:
4815       \int_zero:N \l_@@_local_start_int
4816     }
4817   }
4818 }
4819 \int_compare:nNnT \l_@@_local_start_int > 0
4820 {
4821   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
4822   \@@_vline_ii:
```

```

4823     }
4824 }

4825 \cs_new_protected:Npn \@@_test_in_corner_v:
4826 {
4827     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
4828     {
4829         \seq_if_in:NxT
4830         \l_@@_corners_cells_seq
4831         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
4832         { \bool_set_false:N \g_tmpa_bool }
4833     }
4834     {
4835         \seq_if_in:NxT
4836         \l_@@_corners_cells_seq
4837         { \l_tmpa_tl - \l_tmpb_tl }
4838         {
4839             \int_compare:nNnTF \l_tmpb_tl = 1
4840             { \bool_set_false:N \g_tmpa_bool }
4841             {
4842                 \seq_if_in:NxT
4843                 \l_@@_corners_cells_seq
4844                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
4845                 { \bool_set_false:N \g_tmpa_bool }
4846             }
4847         }
4848     }
4849 }

4850 \cs_new_protected:Npn \@@_vline_ii:
4851 {
4852     \bool_set_false:N \l_@@_dotted_bool
4853     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
4854     \bool_if:NTF \l_@@_dotted_bool
4855     \@@_vline_iv:
4856     {
4857         \tl_if_empty:NTF \l_@@_tikz_rule_tl
4858         \@@_vline_iii:
4859         \@@_vline_v:
4860     }
4861 }

```

First the case of a standard rule, that is to say a rule which is not dotted (and the user has not used the key tikz).

```

4862 \cs_new_protected:Npn \@@_vline_iii:
4863 {
4864     \pgfpicture
4865     \pgfrememberpicturepositiononpagetrue
4866     \pgf@relevantforpicturesizefalse
4867     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4868     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4869     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4870     \dim_set_eq:NN \l_tmpb_dim \pgf@x
4871     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4872     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4873     \bool_lazy_all:nT
4874     {
4875         { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
4876         { \cs_if_exist_p:N \CT@drsc@ }
4877         { ! \tl_if_blank_p:V \CT@drsc@ }
4878     }

```

```

4879 {
4880   \group_begin:
4881   \CT@drsc@
4882   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4883   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
4884   \dim_set:Nn \l_@@_tmpd_dim
4885   {
4886     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
4887     * ( \l_@@_multiplicity_int - 1 )
4888   }
4889   \pgfpathrectanglecorners
4890   { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4891   { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
4892   \pgfusepath { fill }
4893   \group_end:
4894 }
4895 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4896 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
4897 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
4898 {
4899   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4900   \dim_sub:Nn \l_tmpb_dim \doublerulesep
4901   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4902   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
4903 }
4904 \CT@arc@
4905 \pgfsetlinewidth { 1.1 \arrayrulewidth }
4906 \pgfsetrectcap
4907 \pgfusepathqstroke
4908 \endpgfpicture
4909 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

4910 \cs_new_protected:Npn \@@_vline_iv:
4911 {
4912   \pgfpicture
4913   \pgfrememberpicturepositiononpagetrue
4914   \pgf@relevantforpicturesizefalse
4915   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4916   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4917   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4918   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4919   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4920   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4921   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4922   \CT@arc@
4923   \@@_draw_line:
4924   \endpgfpicture
4925 }

```

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`).

```

4926 \cs_new_protected:Npn \@@_vline_v:
4927 {
4928   \begin {tikzpicture}
4929   \pgfrememberpicturepositiononpagetrue
4930   \pgf@relevantforpicturesizefalse
4931   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4932   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4933   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4934   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
4935   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }

```

```

4936 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4937 \exp_args:N \tikzset \l_@@_tikz_rule_tl
4938 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
4939 ( \l_tmpb_dim , \l_tmpa_dim ) --
4940 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
4941 \end { tikzpicture }
4942 }

```

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).

```

4943 \cs_new_protected:Npn \@@_draw_vlines:
4944 {
4945   \int_step_inline:nnn
4946   {
4947     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4948     1 2
4949   }
4950   {
4951     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4952     { \int_eval:n { \c@jCol + 1 } }
4953     \c@jCol
4954   }
4955   {
4956     \tl_if_eq:NnF \l_@@_vlines_clist { all }
4957     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4958     { \@@_vline:n { position = ##1 } }
4959   }
4960 }

```

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}`.

```

4961 \cs_new_protected:Npn \@@_hline:n #1
4962 {

```

The group is for the options.

```

4963   \group_begin:
4964   \int_zero_new:N \l_@@_end_int
4965   \int_set_eq:NN \l_@@_end_int \c@jCol
4966   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
4967   \@@_hline_i:
4968   \group_end:
4969 }

```

```

4970 \cs_new_protected:Npn \@@_hline_i:
4971 {
4972   \int_zero_new:N \l_@@_local_start_int
4973   \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`.

```

4974   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
4975   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
4976   \l_tmpb_tl
4977   {

```

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.

```

4978   \bool_gset_true:N \g_tmpa_bool
4979   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq

```

```

4980     { \@@_test_hline_in_block:nnnnn ##1 }
4981     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4982     { \@@_test_hline_in_block:nnnnn ##1 }
4983     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4984     { \@@_test_hline_in_stroken_block:nnnn ##1 }
4985     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4986     \bool_if:NTF \g_tmpa_bool
4987     {
4988         \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.

```

4989         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
4990     }
4991     {
4992         \int_compare:nNnT \l_@@_local_start_int > 0
4993         {
4994             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
4995             \@@_hline_ii:
4996             \int_zero:N \l_@@_local_start_int
4997         }
4998     }
4999 }
5000 \int_compare:nNnT \l_@@_local_start_int > 0
5001 {
5002     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5003     \@@_hline_ii:
5004 }
5005 }

```

```

5006 \cs_new_protected:Npn \@@_test_in_corner_h:
5007 {
5008     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5009     {
5010         \seq_if_in:NxT
5011         \l_@@_corners_cells_seq
5012         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5013         { \bool_set_false:N \g_tmpa_bool }
5014     }
5015     {
5016         \seq_if_in:NxT
5017         \l_@@_corners_cells_seq
5018         { \l_tmpa_tl - \l_tmpb_tl }
5019         {
5020             \int_compare:nNnTF \l_tmpa_tl = 1
5021             { \bool_set_false:N \g_tmpa_bool }
5022             {
5023                 \seq_if_in:NxT
5024                 \l_@@_corners_cells_seq
5025                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5026                 { \bool_set_false:N \g_tmpa_bool }
5027             }
5028         }
5029     }
5030 }

```

```

5031 \cs_new_protected:Npn \@@_hline_ii:
5032 {
5033     \bool_set_false:N \l_@@_dotted_bool
5034     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5035     \bool_if:NTF \l_@@_dotted_bool
5036     \@@_hline_iv:

```

```

5037     {
5038         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5039             \@@_hline_iii:
5040             \@@_hline_v:
5041     }
5042 }

```

First the case of a standard rule, that is to say a rule which is not dotted.

```

5043 \cs_new_protected:Npn \@@_hline_iii:
5044 {
5045     \pgfpicture
5046     \pgfrememberpicturepositiononpagetrue
5047     \pgf@relevantforpicturesizefalse
5048     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5049     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5050     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5051     \dim_set_eq:NN \l_tmpb_dim \pgf@y
5052     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5053     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5054     \bool_lazy_all:nT
5055     {
5056         { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5057         { \cs_if_exist_p:N \CT@drsc@ }
5058         { ! \tl_if_blank_p:V \CT@drsc@ }
5059     }
5060     {
5061         \group_begin:
5062         \CT@drsc@
5063         \dim_set:Nn \l_@@_tmpd_dim
5064         {
5065             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5066             * ( \l_@@_multiplicity_int - 1 )
5067         }
5068         \pgfpathrectanglecorners
5069         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5070         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5071         \pgfusepathqfill
5072         \group_end:
5073     }
5074     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5075     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5076     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5077     {
5078         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5079         \dim_sub:Nn \l_tmpb_dim \doublerulesep
5080         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5081         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5082     }
5083     \CT@arc@
5084     \pgfsetlinewidth { 1.1 \arrayrulewidth }
5085     \pgfsetrectcap
5086     \pgfusepathqstroke
5087     \endpgfpicture
5088 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (\hline doesn't).


```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

```

5089 \cs_new_protected:Npn \@@_hline_iv:
5090 {
5091   \pgfpicture
5092   \pgfrememberpicturepositiononpagetrue
5093   \pgf@relevantforpicturesizefalse
5094   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5095   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5096   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5097   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5098   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5099   \int_compare:nNnT \l_@@_local_start_int = 1
5100   {
5101     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5102     \bool_if:NT \l_@@_NiceArray_bool
5103     { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by `0.5 \l_@@_inter_dots_dim` is *ad hoc* for a better result.

```

5104     \tl_if_eq:NnF \g_@@_left_delim_tl (
5105       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
5106     )
5107     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5108     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5109     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5110     {
5111       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5112       \bool_if:NT \l_@@_NiceArray_bool
5113       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5114       \tl_if_eq:NnF \g_@@_right_delim_tl )
5115       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
5116     }
5117     \CT@arc@
5118     \@@_draw_line:
5119     \endpgfpicture
5120   }

```

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`).

```

5121 \cs_new_protected:Npn \@@_hline_v:
5122 {
5123   \begin { tikzpicture }
5124   \pgfrememberpicturepositiononpagetrue
5125   \pgf@relevantforpicturesizefalse
5126   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5127   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5128   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5129   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }

```

```

5130 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5131 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5132 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5133 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5134 ( \l_tmpa_dim , \l_tmpb_dim ) --
5135 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5136 \end { tikzpicture }
5137 }

```

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)).

```

5138 \cs_new_protected:Npn \@@_draw_hlines:
5139 {
5140   \int_step_inline:nnn
5141   {
5142     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5143     1 2
5144   }
5145   {
5146     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5147     { \int_eval:n { \c@iRow + 1 } }
5148     \c@iRow
5149   }
5150   {
5151     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5152     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5153     { \@@_hline:n { position = ##1 } }
5154   }
5155 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5156 \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.

```

5157 \cs_set:Npn \@@_Hline_i:n #1
5158 {
5159   \peek_meaning_ignore_spaces:NTF \Hline
5160   { \@@_Hline_ii:nn { #1 + 1 } }
5161   { \@@_Hline_iii:n { #1 } }
5162 }
5163 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5164 \cs_set:Npn \@@_Hline_iii:n #1
5165 {
5166   \skip_vertical:n
5167   {
5168     \arrayrulewidth * ( #1 )
5169     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
5170   }
5171   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5172   {
5173     \@@_hline:n
5174     {
5175       position = \int_eval:n { \c@iRow + 1 } ,
5176       multiplicity = #1
5177     }
5178   }
5179   \ifnum 0 = ` { \fi }
5180 }

```

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).

```
5181 \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` in `\NiceMatrixOptions`).

```
5182 \cs_new_protected:Npn \@@_custom_line:n #1
5183 {
5184   \str_clear_new:N \l_@@_command_str
5185   \str_clear_new:N \l_@@_letter_str
5186   \dim_zero_new:N \l_@@_rule_width_dim
```

The token list `\l_tmpa_tl` is for the key `color`.

```
5187   \tl_clear:N \l_tmpa_tl
```

The flag `\l_tmpa_bool` will indicate whether the key `tikz` is present.

```
5188   \bool_set_false:N \l_tmpa_bool
```

The flag `\l_tmpb_bool` will indicate whether the key `width` is present.

```
5189   \bool_set_false:N \l_tmpb_bool
5190   \keys_set_known:nn { NiceMatrix / Custom-Line } { #1 }
5191   \bool_if:NT \l_tmpa_bool
5192   {
```

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.

```
5193     \cs_if_exist:NF \tikzpicture
5194     { \@@_error:n { tikz~in~custom~line~without~tikz } }
5195     \tl_if_empty:NF \l_tmpa_tl
5196     { \@@_error:n { color~in~custom~line~with~tikz } }
5197   }
5198   \bool_if:NT \l_tmpb_bool
5199   {
5200     \bool_if:NF \l_tmpa_bool
5201     { \@@_error:n { key~width~without~key~tikz } }
5202   }
```

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.

```
5203   \bool_lazy_and:nnTF
5204   { \str_if_empty_p:N \l_@@_letter_str }
5205   { \str_if_empty_p:N \l_@@_command_str }
5206   { \@@_error:n { No~letter~and~no~command } }
5207   {
5208     \str_if_empty:NF \l_@@_letter_str
5209     {
5210       \int_compare:nNnTF { \str_count:N \l_@@_letter_str } = 1
5211       {
5212         \exp_args:NnV \tl_if_in:NnTF
5213         \c_@@_forbidden_letters_str \l_@@_letter_str
5214         { \@@_error:n { Forbidden~letter } }
5215       }
5216       \exp_args:Nnx \keys_define:nn { NiceMatrix / ColumnTypes }
5217       {
```

```

5218         \l_@@_letter_str .code:n =
5219         { \@@_custom_line_i:n { \exp_not:n { #1 } } }
5220     }
5221 }
5222 }
5223 { \@@_error:n { Several-letters } }
5224 }
5225 \str_if_empty:NF \l_@@_command_str
5226 {

```

The flag `\l_tmpa_bool` means that the key 'tikz' have been used. When the key 'tikz' has not been used, the width of the rule is computed with the multiplicity of the rule.

```

5227     \bool_if:NF \l_tmpa_bool
5228     {
5229         \dim_set:Nn \l_@@_rule_width_dim
5230         {
5231             \arrayrulewidth * \l_@@_tmpc_int
5232             + \doublerulesep * ( \l_@@_tmpc_int - 1 )
5233         }
5234     }
5235     \exp_args:NnV \@@_define_h_custom_line:nn
5236     { #1 }
5237     \l_@@_rule_width_dim
5238 }
5239 }
5240 }
5241 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX:|()[]!@<> }

```

The previous command `\@@_custom_line: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 only entries for a few keys.

```

5242 \keys_define:nn { NiceMatrix / Custom-Line }
5243 {
5244     % here, we will use change in the future to use .tl_set:N
5245     letter .code:n = \str_set:Nn \l_@@_letter_str { #1 } ,
5246     letter .value_required:n = true ,
5247     % here, we will use change in the future to use .tl_set:N
5248     command .code:n = \str_set:Nn \l_@@_command_str { #1 } ,
5249     command .value_required:n = true ,
5250     multiplicity .int_set:N = \l_@@_tmpc_int ,
5251     multiplicity .initial:n = 1 ,
5252     multiplicity .value_required:n = true ,
5253     color .tl_set:N = \l_tmpa_tl ,
5254     color .value_required:n = true ,

```

When the key `tikz` is used, the rule will be drawn with Tikz by using the set of keys specified in the value of that key `tikz`.

```

5255     tikz .code:n = \bool_set_true:N \l_tmpa_bool ,

```

The key `width` must be used only when the key `tikz` is used. When used, the key `width` specifies the width of the rule: it will be used to reserve space (in the preamble of the array or in the command for the horizontal rules).

```

5256     width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5257     \bool_set_true:N \l_tmpb_bool ,
5258     width .value_required:n = true ,
5259 }

```

The following command will create the command that the final user will use in its array to draw a horizontal rule (hence the 'h' in the name). `#1` is the whole set of keys to pass to `\@@_line:n` and `#2` is the width of the whole rule.

```

5260 \cs_new_protected:Npn \@@_define_h_custom_line:nn #1 #2
5261 {

```

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`.

```

5262   \cs_set:cpn \l_@@_command_str
5263   {
5264     \noalign
5265     {
5266       \skip_vertical:n { #2 }
5267       \tl_gput_right:Nx \g_@@_internal_code_after_tl
5268       { \@@_hline:n { #1 , position = \int_eval:n { \c@iRow + 1 } } }
5269     }
5270   }
5271 }

```

The flag `\l_tmpa_bool` means that the key 'tikz' have been used. When the key 'tikz' has not been used, the width of the rule is computed with the multiplicity of the rule.

```

5272 \cs_new_protected:Npn \@@_custom_line_i:n #1
5273 {
5274   \bool_if:NF \l_tmpa_bool
5275   {
5276     \dim_set:Nn \l_@@_rule_width_dim
5277     {
5278       \arrayrulewidth * \l_@@_tmpc_int
5279       + \doublerulesep * ( \l_@@_tmpc_int - 1 )
5280     }
5281   }
5282   \tl_gput_right:Nx \g_@@_preamble_tl
5283   {
5284     \exp_not:N !
5285     { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } }
5286   }
5287   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5288   { \@@_vline:n { #1 , position = \int_eval:n { \c@jCol + 1 } } }
5289 }

```

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.

```

5290 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4
5291 {
5292   \bool_lazy_all:nT
5293   {
5294     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
5295     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5296     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5297     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5298   }
5299   { \bool_gset_false:N \g_tmpa_bool }
5300 }

```

The same for vertical rules.

```

5301 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4
5302 {
5303   \bool_lazy_all:nT
5304   {
5305     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5306     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5307     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
5308     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5309   }
5310   { \bool_gset_false:N \g_tmpa_bool }
5311 }

```

```

5312 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
5313 {
5314   \bool_lazy_all:nT
5315   {
5316     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5317     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
5318     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5319     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5320   }
5321   { \bool_gset_false:N \g_tmpa_bool }
5322 }
5323 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
5324 {
5325   \bool_lazy_all:nT
5326   {
5327     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5328     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5329     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5330     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
5331   }
5332   { \bool_gset_false:N \g_tmpa_bool }
5333 }

```

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.

```

5334 \cs_new_protected:Npn \@@_compute_corners:
5335 {

```

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.

```

5336   \seq_clear_new:N \l_@@_corners_cells_seq
5337   \clist_map_inline:Nn \l_@@_corners_clist
5338   {
5339     \str_case:nnF { ##1 }
5340     {
5341       { NW }
5342       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
5343       { NE }
5344       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
5345       { SW }
5346       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
5347       { SE }
5348       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
5349     }
5350     { \@@_error:nn { bad~corner } { ##1 } }
5351   }

```

Even if the user has used the `key corners` the list of cells in the corners may be empty.

```

5352   \seq_if_empty:NF \l_@@_corners_cells_seq
5353   {

```

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.

```

5354   \tl_gput_right:Nx \g_@@_aux_tl
5355   {
5356     \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
5357     { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
5358   }

```

```

5359     }
5360 }

```

“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.

```

5361 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
5362 {

```

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.

```

5363   \bool_set_false:N \l_tmpa_bool
5364   \int_zero_new:N \l_@@_last_empty_row_int
5365   \int_set:Nn \l_@@_last_empty_row_int { #1 }
5366   \int_step_inline:nnnn { #1 } { #3 } { #5 }
5367   {
5368     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
5369     \bool_lazy_or:nnTF
5370     {
5371       \cs_if_exist_p:c
5372       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
5373     }
5374     \l_tmpb_bool
5375     { \bool_set_true:N \l_tmpa_bool }
5376     {
5377       \bool_if:NF \l_tmpa_bool
5378       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
5379     }
5380   }

```

Now, you determine the last empty cell in the row of number 1.

```

5381   \bool_set_false:N \l_tmpa_bool
5382   \int_zero_new:N \l_@@_last_empty_column_int
5383   \int_set:Nn \l_@@_last_empty_column_int { #2 }
5384   \int_step_inline:nnnn { #2 } { #4 } { #6 }
5385   {
5386     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
5387     \bool_lazy_or:nnTF
5388     \l_tmpb_bool
5389     {
5390       \cs_if_exist_p:c
5391       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
5392     }
5393     { \bool_set_true:N \l_tmpa_bool }
5394     {
5395       \bool_if:NF \l_tmpa_bool
5396       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
5397     }
5398   }

```

Now, we loop over the rows.

```

5399   \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
5400   {

```

We treat the row number ##1 with another loop.

```

5401     \bool_set_false:N \l_tmpa_bool
5402     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
5403     {
5404         \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
5405         \bool_lazy_or:nnTF
5406             \l_tmpb_bool
5407             {
5408                 \cs_if_exist_p:c
5409                     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
5410             }
5411         { \bool_set_true:N \l_tmpa_bool }
5412         {
5413             \bool_if:NF \l_tmpa_bool
5414             {
5415                 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
5416                 \seq_put_right:Nn
5417                     \l_@@_corners_cells_seq
5418                     { ##1 - #####1 }
5419             }
5420         }
5421     }
5422 }
5423 }
```

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`).

```

5424 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
5425 {
5426     \int_set:Nn \l_tmpa_int { #1 }
5427     \int_set:Nn \l_tmpb_int { #2 }
5428     \bool_set_false:N \l_tmpb_bool
5429     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5430         { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
5431 }
5432 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
5433 {
5434     \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
5435     {
5436         \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
5437         {
5438             \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
5439             {
5440                 \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
5441                 { \bool_set_true:N \l_tmpb_bool }
5442             }
5443         }
5444     }
5445 }
```

The commands to draw dotted lines to separate columns and rows

These commands don't use the normal nodes, the medium nor the large nodes. They only use the `col` nodes and the `row` nodes.

Horizontal dotted lines

The following command must *not* be protected because it's meant to be expanded in a `\noalign`.

```

5446 \cs_new:Npn \@@_hdottedline:
5447 {
```



```

5448 \noalign { \skip_vertical:N 2\l_@@_radius_dim }
5449 \@@_hdottedline_i:
5450 }

```

On the other side, the following command should be protected.

```

5451 \cs_new_protected:Npn \@@_hdottedline_i:
5452 {

```

We write in the internal `\CodeAfter` the instruction that will actually draw the dotted line. It's not possible to draw this dotted line now because we don't know the length of the line (we don't even know the number of columns).

```

5453 \tl_gput_right:Nx \g_@@_internal_code_after_tl
5454 { \@@_hdottedline:n { \int_use:N \c@iRow } }
5455 }

```

The command `\@@_hdottedline:n` is the command written in the internal `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```

5456 \cs_new_protected:Npn \@@_hdottedline:n #1
5457 { \@@_hline:n { position = #1 , end = \int_use:N \c@jCol , dotted } }

```

Vertical dotted lines

```

5458 \cs_new_protected:Npn \@@_vdottedline:n #1
5459 { \@@_vline:n { position = \int_eval:n { #1 + 1 } , dotted } }

```

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.

```

5460 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

5461 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5462 {
5463   auto-columns-width .code:n =
5464   {
5465     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5466     \dim_gzero_new:N \g_@@_max_cell_width_dim
5467     \bool_set_true:N \l_@@_auto_columns_width_bool
5468   }
5469 }

5470 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
5471 {
5472   \int_gincr:N \g_@@_NiceMatrixBlock_int
5473   \dim_zero:N \l_@@_columns_width_dim
5474   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5475   \bool_if:NT \l_@@_block_auto_columns_width_bool
5476   {
5477     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5478     {
5479       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
5480       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5481     }
5482   }
5483 }

```

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`).

```

5484 {
5485   \bool_if:NT \l_@@_block_auto_columns_width_bool
5486   {
5487     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5488     \iow_shipout:Nx \@mainaux
5489     {
5490       \cs_gset:cpn
5491       { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
5492       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5493     }
5494     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5495   }
5496 }
```

The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

5497 \cs_generate_variant:Nn \dim_min:nn { v n }
5498 \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).

```

5499 \cs_new_protected:Npn \@@_create_extra_nodes:
5500 {
5501   \bool_if:nTF \l_@@_medium_nodes_bool
5502   {
5503     \bool_if:NTF \l_@@_large_nodes_bool
5504     \@@_create_medium_and_large_nodes:
5505     \@@_create_medium_nodes:
5506   }
5507   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5508 }
```

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`.

```

5509 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5510 {
5511   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
```

```

5512 {
5513   \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
5514   \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
5515   \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
5516   \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
5517 }
5518 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5519 {
5520   \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
5521   \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
5522   \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
5523   \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
5524 }

```

We begin the two nested loops over the rows and the columns of the array.

```

5525 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5526 {
5527   \int_step_variable:nnNn
5528     \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.

```

5529 {
5530   \cs_if_exist:cT
5531     { 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.

```

5532 {
5533   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5534   \dim_set:cn { l_@@_row\_@@_i: _min_dim }
5535   { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
5536   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5537   {
5538     \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
5539     { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
5540   }

```

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.

```

5541   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
5542   \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
5543   { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
5544   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5545   {
5546     \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
5547     { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
5548   }
5549 }
5550 }
5551 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

5552 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5553 {
5554   \dim_compare:nnNt
5555     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
5556   {
5557     \@@_qpoint:n { row - \@@_i: - base }
5558     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
5559     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
5560   }
5561 }
5562 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

```

5563 {
5564   \dim_compare:nNnT
5565     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
5566     {
5567       \@@_qpoint:n { col - \@@_j: }
5568       \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
5569       \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
5570     }
5571   }
5572 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

5573 \cs_new_protected:Npn \@@_create_medium_nodes:
5574 {
5575   \pgfpicture
5576   \pgfrememberpicturepositiononpagetrue
5577   \pgf@relevantforpicturesizefalse
5578   \@@_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”.

```

5579   \tl_set:Nn \l_@@_suffix_tl { -medium }
5580   \@@_create_nodes:
5581   \endpgfpicture
5582 }

```

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:`.

```

5583 \cs_new_protected:Npn \@@_create_large_nodes:
5584 {
5585   \pgfpicture
5586   \pgfrememberpicturepositiononpagetrue
5587   \pgf@relevantforpicturesizefalse
5588   \@@_computations_for_medium_nodes:
5589   \@@_computations_for_large_nodes:
5590   \tl_set:Nn \l_@@_suffix_tl { - large }
5591   \@@_create_nodes:
5592   \endpgfpicture
5593 }
5594 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5595 {
5596   \pgfpicture
5597   \pgfrememberpicturepositiononpagetrue
5598   \pgf@relevantforpicturesizefalse
5599   \@@_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”.

```

5600   \tl_set:Nn \l_@@_suffix_tl { - medium }
5601   \@@_create_nodes:
5602   \@@_computations_for_large_nodes:
5603   \tl_set:Nn \l_@@_suffix_tl { - large }
5604   \@@_create_nodes:
5605   \endpgfpicture
5606 }

```

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.

```

5607 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5608 {
5609   \int_set:Nn \l_@@_first_row_int 1
5610   \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`.

```

5611   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5612   {
5613     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5614     {
5615       (
5616         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5617         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
5618       )
5619       / 2
5620     }
5621     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
5622     { l_@@_row _ \@@_i: _ min _ dim }
5623   }
5624   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5625   {
5626     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5627     {
5628       (
5629         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5630         \dim_use:c
5631         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
5632       )
5633       / 2
5634     }
5635     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
5636     { l_@@_column _ \@@_j: _ max _ dim }
5637   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

5638   \dim_sub:cn
5639   { l_@@_column _ 1 _ min _ dim }
5640   \l_@@_left_margin_dim
5641   \dim_add:cn
5642   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5643   \l_@@_right_margin_dim
5644 }

```

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).

```

5645 \cs_new_protected:Npn \@@_create_nodes:
5646 {
5647   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5648   {
5649     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5650     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

5651       \@@_pgf_rect_node:nnnnn
5652       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5653       { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }

```

```

5654         { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
5655         { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
5656         { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
5657     \str_if_empty:NF \l_@@_name_str
5658     {
5659         \pgfnodealias
5660         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5661         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5662     }
5663 }
5664 }

```

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 .

```

5665     \seq_mapthread_function:NNN
5666     \g_@@_multicolumn_cells_seq
5667     \g_@@_multicolumn_sizes_seq
5668     \@@_node_for_multicolumn:nn
5669 }

5670 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5671 {
5672     \cs_set_nopar:Npn \@@_i: { #1 }
5673     \cs_set_nopar:Npn \@@_j: { #2 }
5674 }

```

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”).

```

5675 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5676 {
5677     \@@_extract_coords_values: #1 \q_stop
5678     \@@_pgf_rect_node:nnnnn
5679     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5680     { \dim_use:c { l_@@_column_ \@@_j: _min _dim } }
5681     { \dim_use:c { l_@@_row_ \@@_i: _min _dim } }
5682     { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max _dim } }
5683     { \dim_use:c { l_@@_row_ \@@_i: _max _dim } }
5684     \str_if_empty:NF \l_@@_name_str
5685     {
5686         \pgfnodealias
5687         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5688         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
5689     }
5690 }

```

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.

```

5691 \keys_define:nn { NiceMatrix / Block / FirstPass }
5692 {
5693     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5694     l .value_forbidden:n = true ,
5695     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5696     r .value_forbidden:n = true ,

```

```

5697 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5698 c .value_forbidden:n = true ,
5699 L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5700 L .value_forbidden:n = true ,
5701 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5702 R .value_forbidden:n = true ,
5703 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5704 C .value_forbidden:n = true ,
5705 t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5706 t .value_forbidden:n = true ,
5707 b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5708 b .value_forbidden:n = true ,
5709 color .tl_set:N = \l_@@_color_tl ,
5710 color .value_required:n = true ,
5711 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
5712 respect-arraystretch .default:n = true ,
5713 }

```

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.

```

5714 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
5715 {

```

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).

```

5716 \peek_remove_spaces:n
5717 {
5718   \tl_if_blank:nTF { #2 }
5719     { \@@_Block_i 1-1 \q_stop }
5720     { \@@_Block_i #2 \q_stop }
5721     { #1 } { #3 } { #4 }
5722   }
5723 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5724 \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.

```

5725 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5726 {

```

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).

```

5727 \bool_lazy_or:nnTF
5728   { \tl_if_blank_p:n { #1 } }
5729   { \str_if_eq_p:nn { #1 } { * } }
5730   { \int_set:Nn \l_tmpa_int { 100 } }
5731   { \int_set:Nn \l_tmpa_int { #1 } }
5732 \bool_lazy_or:nnTF
5733   { \tl_if_blank_p:n { #2 } }
5734   { \str_if_eq_p:nn { #2 } { * } }
5735   { \int_set:Nn \l_tmpb_int { 100 } }
5736   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5737 \int_compare:nNnTF \l_tmpb_int = 1
5738 {
5739   \str_if_empty:NTF \l_@@_hpos_cell_str
5740   { \str_set:Nn \l_@@_hpos_block_str c }
5741   { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
5742 }
5743 { \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.

```

5744 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
5745 \tl_set:Nx \l_tmpa_tl
5746 {
5747   { \int_use:N \c@iRow }
5748   { \int_use:N \c@jCol }
5749   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5750   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5751 }

```

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).

```

5752 \bool_if:nTF
5753 {
5754   (
5755     \int_compare_p:nNn { \l_tmpa_int } = 1
5756     ||
5757     \int_compare_p:nNn { \l_tmpb_int } = 1
5758   )
5759   && ! \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.

```

5760   && ! \l_@@_X_column_bool
5761 }
5762 { \exp_args:Nxx \@@_Block_iv:nnnnn }
5763 { \exp_args:Nxx \@@_Block_v:nnnnn }
5764 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5765 }

```

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).

```

5766 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5767 {
5768   \int_gincr:N \g_@@_block_box_int
5769   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5770   {
5771     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5772     {
5773       \@@_actually_diagbox:nnnnnn
5774       { \int_use:N \c@iRow }

```



```

5775         { \int_use:N \c@jCol }
5776         { \int_eval:n { \c@iRow + #1 - 1 } }
5777         { \int_eval:n { \c@jCol + #2 - 1 } }
5778         { \exp_not:n { ##1 } } } { \exp_not:n { ##2 } }
5779     }
5780 }
5781 \box_gclear_new:c
5782 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5783 \hbox_gset:cn
5784 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5785 {

```

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}`).

```

5786     \tl_if_empty:NTF \l_@@_color_tl
5787     { \int_compare:nNnT { #2 } = 1 \set@color }
5788     { \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`.

```

5789     \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5790     \group_begin:
5791     \bool_if:NF \l_@@_respect_arraystretch_bool
5792     { \cs_set:Npn \arraystretch { 1 } }
5793     \dim_zero:N \extrarowheight
5794     #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).

```

5795     \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
5796     \bool_if:NTF \l_@@_NiceTabular_bool
5797     {
5798         \bool_lazy_all:nTF
5799         {
5800             { \int_compare_p:nNn { #2 } = 1 }
5801             { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
5802             { ! \l_@@_respect_arraystretch_bool }
5803         }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

5804     {
5805         \begin { minipage } [ \l_@@_vpos_of_block_tl ]
5806         { \l_@@_col_width_dim }
5807         \str_case:Nn \l_@@_hpos_block_str
5808         {
5809             c \centering
5810             r \raggedleft
5811             l \raggedright
5812         }
5813         #5
5814         \end { minipage }
5815     }
5816     {
5817         \use:x
5818         {
5819             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5820             { @ { } \l_@@_hpos_block_str @ { } }
5821         }

```

```

5822         #5
5823     \end { tabular }
5824 }
5825 }
5826 {
5827     \c_math_toggle_token
5828     \use:x
5829     {
5830         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5831         { @ { } \l_@@_hpos_block_str @ { } }
5832     }
5833     #5
5834     \end { array }
5835     \c_math_toggle_token
5836 }
5837 \group_end:
5838 }
5839 \bool_if:NT \g_@@_rotate_bool
5840 {
5841     \box_grotate:cn
5842     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5843     { 90 }
5844     \bool_gset_false:N \g_@@_rotate_bool
5845 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5846 \int_compare:nNnT { #2 } = 1
5847 {
5848     \dim_gset:Nn \g_@@_blocks_wd_dim
5849     {
5850         \dim_max:nn
5851         \g_@@_blocks_wd_dim
5852         {
5853             \box_wd:c
5854             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5855         }
5856     }
5857 }

```

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.

```

5858 \int_compare:nNnT { #1 } = 1
5859 {
5860     \dim_gset:Nn \g_@@_blocks_ht_dim
5861     {
5862         \dim_max:nn
5863         \g_@@_blocks_ht_dim
5864         {
5865             \box_ht:c
5866             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5867         }
5868     }
5869     \dim_gset:Nn \g_@@_blocks_dp_dim
5870     {
5871         \dim_max:nn
5872         \g_@@_blocks_dp_dim
5873         {
5874             \box_dp:c
5875             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5876         }
5877     }
5878 }
5879 \seq_gput_right:Nx \g_@@_blocks_seq

```

```

5880 {
5881   \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.

```

5882   { \exp_not:n { #3 } , \l_@@_hpos_block_str }
5883   {
5884     \box_use_drop:c
5885     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5886   }
5887 }
5888 }

```

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.

```

5889 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5890 {
5891   \seq_gput_right:Nx \g_@@_blocks_seq
5892   {
5893     \l_tmpa_tl
5894     { \exp_not:n { #3 } }
5895     \exp_not:n
5896     {
5897       {
5898         \bool_if:NTF \l_@@_NiceTabular_bool
5899         {
5900           \group_begin:
5901           \bool_if:NF \l_@@_respect_arraystretch_bool
5902             { \cs_set:Npn \arraystretch { 1 } }
5903           \dim_zero:N \extrarowheight
5904           #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).

```

5905     \bool_if:NT \g_@@_rotate_bool
5906     { \str_set:Nn \l_@@_hpos_block_str c }
5907     \use:x
5908     {
5909       \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5910       { @ { } \l_@@_hpos_block_str @ { } }
5911     }
5912     #5
5913   \end { tabular }
5914   \group_end:
5915 }
5916 {
5917   \group_begin:
5918   \bool_if:NF \l_@@_respect_arraystretch_bool
5919     { \cs_set:Npn \arraystretch { 1 } }
5920   \dim_zero:N \extrarowheight
5921   #4
5922   \bool_if:NT \g_@@_rotate_bool
5923     { \str_set:Nn \l_@@_hpos_block_str c }
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     \group_end:
5934 }
5935 }
5936 }
5937 }
5938 }

```

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).

```

5939 \keys_define:nn { NiceMatrix / Block / SecondPass }
5940 {
5941     tikz .code:n =
5942         \bool_if:NTF \c_@@_tikz_loaded_bool
5943         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
5944         { \@@_error:n { tikz-key-without~tikz } } ,
5945     tikz .value_required:n = true ,
5946     fill .tl_set:N = \l_@@_fill_tl ,
5947     fill .value_required:n = true ,
5948     draw .tl_set:N = \l_@@_draw_tl ,
5949     draw .default:n = default ,
5950     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5951     rounded-corners .default:n = 4 pt ,
5952     color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5953     color .value_required:n = true ,
5954     borders .clist_set:N = \l_@@_borders_clist ,
5955     borders .value_required:n = true ,
5956     hvlines .meta:n = { vlines , hlines } ,
5957     vlines .bool_set:N = \l_@@_vlines_block_bool ,
5958     vlines .default:n = true ,
5959     hlines .bool_set:N = \l_@@_hlines_block_bool ,
5960     hlines .default:n = true ,
5961     line-width .dim_set:N = \l_@@_line_width_dim ,
5962     line-width .value_required:n = true ,
5963     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5964     l .value_forbidden:n = true ,
5965     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5966     r .value_forbidden:n = true ,
5967     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5968     c .value_forbidden:n = true ,
5969     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
5970         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5971     L .value_forbidden:n = true ,
5972     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
5973         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5974     R .value_forbidden:n = true ,
5975     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
5976         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5977     C .value_forbidden:n = true ,
5978     t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5979     t .value_forbidden:n = true ,
5980     b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5981     b .value_forbidden:n = true ,
5982     name .tl_set:N = \l_@@_block_name_str ,
5983     name .value_required:n = true ,
5984     name .initial:n = ,
5985     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,

```

```

5986   respect-arraystretch .default:n = true ,
5987   v-center .bool_set:N = \l_@@_v_center_bool ,
5988   v-center .default:n = true ,
5989   v-center .initial:n = false ,
5990   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
5991 }

```

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.

```

5992 \cs_new_protected:Npn \@@_draw_blocks:
5993 {
5994   \cs_set_eq:NN \ialign \@@_old_ialign:
5995   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
5996 }
5997 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5998 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5999   \int_zero_new:N \l_@@_last_row_int
6000   \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.

```

6001   \int_compare:nNnTF { #3 } > { 99 }
6002   { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
6003     { \int_set:Nn \l_@@_last_row_int { #3 } }
6004   \int_compare:nNnTF { #4 } > { 99 }
6005   { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
6006     { \int_set:Nn \l_@@_last_col_int { #4 } }
6007   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6008   {
6009     \int_compare:nTF
6010     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6011     {
6012       \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
6013       \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
6014       \group_begin:
6015       \globaldefs = 1
6016       \@@_msg_redirect_name:nn { columns~not~used } { none }
6017       \group_end:
6018     }
6019     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
6020   }
6021   {
6022     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6023     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
6024     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6025   }
6026 }
6027 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6028 {

```

The group is for the keys.

```

6029   \group_begin:
6030   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }

```

We restrict the use of the key `v-center` to the case of a mono-row block.

```

6031 \bool_if:NT \l_@@_v_center_bool
6032 {
6033   \int_compare:nNnF { #1 } = { #3 }
6034   {
6035     \@@_error:n { Wrong~use~of~v-center }
6036     \bool_set_false:N \l_@@_v_center_bool
6037   }
6038 }
6039 \bool_if:NT \l_@@_vlines_block_bool
6040 {
6041   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6042   {
6043     \@@_vlines_block:nnn
6044     { \exp_not:n { #5 } }
6045     { #1 - #2 }
6046     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6047   }
6048 }
6049 \bool_if:NT \l_@@_hlines_block_bool
6050 {
6051   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6052   {
6053     \@@_hlines_block:nnn
6054     { \exp_not:n { #5 } }
6055     { #1 - #2 }
6056     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6057   }
6058 }
6059 \bool_if:nT
6060 { ! \l_@@_vlines_block_bool && ! \l_@@_hlines_block_bool }
6061 {

```

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).

```

6062 \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6063 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6064 }
6065 \tl_if_empty:NF \l_@@_draw_tl
6066 {
6067   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6068   {
6069     \@@_stroke_block:nnn
6070     { \exp_not:n { #5 } }
6071     { #1 - #2 }
6072     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6073   }
6074   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6075   { { #1 } { #2 } { #3 } { #4 } }
6076 }
6077 \clist_if_empty:NF \l_@@_borders_clist
6078 {
6079   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6080   {
6081     \@@_stroke_borders_block:nnn
6082     { \exp_not:n { #5 } }
6083     { #1 - #2 }
6084     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6085   }
6086 }
6087 \tl_if_empty:NF \l_@@_fill_tl
6088 {

```

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`.

```

6089     \exp_last_unbraced:NV \@@_extract_brackets \l_@@_fill_tl \q_stop
6090     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6091     {
6092         \exp_not:N \roundedrectanglecolor
6093         [ \l_tmpa_tl ]
6094         { \exp_not:N \l_tmpb_tl }
6095         { #1 - #2 }
6096         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6097         { \dim_use:N \l_@@_rounded_corners_dim }
6098     }
6099 }

6100 \seq_if_empty:NF \l_@@_tikz_seq
6101 {
6102     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6103     {
6104         \@@_block_tikz:nnnnn
6105         { #1 }
6106         { #2 }
6107         { \int_use:N \l_@@_last_row_int }
6108         { \int_use:N \l_@@_last_col_int }
6109         { \seq_use:Nn \l_@@_tikz_seq { , } }
6110     }
6111 }

6112 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6113 {
6114     \tl_gput_right:Nx \g_@@_internal_code_after_tl
6115     {
6116         \@@_actually_diagbox:nnnnnn
6117         { #1 }
6118         { #2 }
6119         { \int_use:N \l_@@_last_row_int }
6120         { \int_use:N \l_@@_last_col_int }
6121         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6122     }
6123 }

6124 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6125 \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.

```

6126 \pgfpicture
6127 \pgfrememberpicturepositiononpagetrue
6128 \pgf@relevantforpicturesizefalse
6129 \@@_qpoint:n { row - #1 }
6130 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6131 \@@_qpoint:n { col - #2 }
6132 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6133 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6134 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6135 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6136 \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.

```

6137 \@@_pgf_rect_node:nnnnn
6138 { \@@_env: - #1 - #2 - block }
6139 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6140 \str_if_empty:NF \l_@@_block_name_str
6141 {
6142 \pgfnodealias
6143 { \@@_env: - \l_@@_block_name_str }
6144 { \@@_env: - #1 - #2 - block }
6145 \str_if_empty:NF \l_@@_name_str
6146 {
6147 \pgfnodealias
6148 { \l_@@_name_str - \l_@@_block_name_str }
6149 { \@@_env: - #1 - #2 - block }
6150 }
6151 }

```

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.

```

6152 \bool_if:NF \l_@@_hpos_of_block_cap_bool
6153 {
6154 \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.

```

6155 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6156 {

```

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.

```

6157 \cs_if_exist:cT
6158 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6159 {
6160 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6161 {
6162 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
6163 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
6164 }
6165 }
6166 }

```

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.

```

6167 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
6168 {
6169 \@@_qpoint:n { col - #2 }
6170 \dim_set_eq:NN \l_tmpb_dim \pgf@x

```



```

6171     }
6172     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
6173     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6174     {
6175         \cs_if_exist:cT
6176         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6177         {
6178             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6179             {
6180                 \pgfpointanchor
6181                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6182                 { east }
6183                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
6184             }
6185         }
6186     }
6187     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
6188     {
6189         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6190         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6191     }
6192     \@@_pgf_rect_node:nnnnn
6193     { \@@_env: - #1 - #2 - block - short }
6194     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6195 }

```

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.

```

6196     \bool_if:NT \l_@@_medium_nodes_bool
6197     {
6198         \@@_pgf_rect_node:nnn
6199         { \@@_env: - #1 - #2 - block - medium }
6200         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
6201         {
6202             \pgfpointanchor
6203             { \@@_env:
6204                 - \int_use:N \l_@@_last_row_int
6205                 - \int_use:N \l_@@_last_col_int - medium
6206             }
6207             { south-east }
6208         }
6209     }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

6210     \bool_if:nTF
6211     { \int_compare_p:nNn { #1 } = { #3 } && ! \l_@@_v_center_bool }
6212     {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

6213         \int_compare:nNnTF { #1 } = 0
6214         { \l_@@_code_for_first_row_tl }
6215         {
6216             \int_compare:nNnT { #1 } = \l_@@_last_row_int
6217             \l_@@_code_for_last_row_tl
6218         }

```

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`.

```

6219         \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

6220         \pgfpointanchor
6221         {

```

```

6222         \@@_env: - #1 - #2 - block
6223         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6224     }
6225     {
6226         \str_case:Vn \l_@@_hpos_block_str
6227         {
6228             c { center }
6229             l { west }
6230             r { east }
6231         }
6232     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

6233     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
6234     \pgfset { inner~sep = \c_zero_dim }
6235     \pgfnode
6236     { rectangle }
6237     {
6238         \str_case:Vn \l_@@_hpos_block_str
6239         {
6240             c { base }
6241             l { base~west }
6242             r { base~east }
6243         }
6244     }
6245     { \box_use_drop:N \l_@@_cell_box } { } { }
6246 }

```

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`).

```

6247     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

6248         \int_compare:nNnT { #2 } = 0
6249         { \str_set:Nn \l_@@_hpos_block_str r }
6250         \bool_if:nT \g_@@_last_col_found_bool
6251         {
6252             \int_compare:nNnT { #2 } = \g_@@_col_total_int
6253             { \str_set:Nn \l_@@_hpos_block_str l }
6254         }
6255         \pgftransformshift
6256         {
6257             \pgfpointanchor
6258             {
6259                 \@@_env: - #1 - #2 - block
6260                 \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6261             }
6262             {
6263                 \str_case:Vn \l_@@_hpos_block_str
6264                 {
6265                     c { center }
6266                     l { west }
6267                     r { east }
6268                 }
6269             }
6270         }
6271         \pgfset { inner~sep = \c_zero_dim }
6272         \pgfnode
6273         { rectangle }
6274         {
6275             \str_case:Vn \l_@@_hpos_block_str
6276             {
6277                 c { center }
6278                 l { west }

```

```

6279         r { east }
6280     }
6281 }
6282 { \box_use_drop:N \l_@@_cell_box } { } { }
6283 }
6284 \endpgfpicture
6285 \group_end:
6286 }

6287 \NewDocumentCommand \@@_extract_brackets { 0 { } }
6288 {
6289     \tl_set:Nn \l_tmpa_tl { #1 }
6290     \@@_store_in_tmpb_tl
6291 }
6292 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
6293 { \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).

```

6294 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
6295 {
6296     \group_begin:
6297     \tl_clear:N \l_@@_draw_tl
6298     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6299     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
6300     \pgfpicture
6301     \pgfrememberpicturepositiononpagetrue
6302     \pgf@relevantforpicturesizefalse
6303     \tl_if_empty:NF \l_@@_draw_tl
6304     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

6305         \str_if_eq:VnTF \l_@@_draw_tl { default }
6306         { \CT@arc@ }
6307         { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
6308     }
6309     \pgfsetcornersarced
6310     {
6311         \pgfpoint
6312         { \dim_use:N \l_@@_rounded_corners_dim }
6313         { \dim_use:N \l_@@_rounded_corners_dim }
6314     }
6315     \@@_cut_on_hyphen:w #2 \q_stop
6316     \bool_lazy_and:nnT
6317     { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
6318     { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
6319     {
6320         \@@_qpoint:n { row - \l_tmpa_tl }
6321         \dim_set:Nn \l_tmpb_dim { \pgf@y }
6322         \@@_qpoint:n { col - \l_tmpb_tl }
6323         \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
6324         \@@_cut_on_hyphen:w #3 \q_stop
6325         \int_compare:nNnT \l_tmpa_tl > \c@iRow
6326         { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
6327         \int_compare:nNnT \l_tmpb_tl > \c@jCol
6328         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
6329         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
6330         \dim_set:Nn \l_tmpa_dim { \pgf@y }
6331         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6332         \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }

```

```

6333 \pgfpathrectanglecorners
6334 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6335 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6336 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

6337 \pgfusepath { stroke }
6338 }
6339 \endpgfpicture
6340 \group_end:
6341 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

6342 \keys_define:nn { NiceMatrix / BlockStroke }
6343 {
6344   color .tl_set:N = \l_@@_draw_tl ,
6345   draw .tl_set:N = \l_@@_draw_tl ,
6346   draw .default:n = default ,
6347   line-width .dim_set:N = \l_@@_line_width_dim ,
6348   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6349   rounded-corners .default:n = 4 pt
6350 }

```

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).

```

6351 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
6352 {
6353   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6354   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6355   \@@_cut_on_hyphen:w #2 \q_stop
6356   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6357   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6358   \@@_cut_on_hyphen:w #3 \q_stop
6359   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6360   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6361   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
6362   {
6363     \use:x
6364     {
6365       \@@_vline:n
6366       {
6367         position = ##1 ,
6368         start = \l_@@_tmpc_tl ,
6369         end = \int_eval:n { \l_tmpa_tl - 1 }
6370       }
6371     }
6372   }
6373 }
6374 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
6375 {
6376   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6377   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6378   \@@_cut_on_hyphen:w #2 \q_stop
6379   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6380   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6381   \@@_cut_on_hyphen:w #3 \q_stop
6382   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6383   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6384   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
6385   {
6386     \use:x
6387     {
6388       \@@_hline:n

```

```

6389         {
6390             position = ##1 ,
6391             start = \l_@@_tmpd_tl ,
6392             end = \int_eval:n { \l_tmpb_tl - 1 }
6393         }
6394     }
6395 }
6396 }

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).

6397 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
6398 {
6399     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6400     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6401     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
6402     { \@@_error:n { borders~forbidden } }
6403     {
6404         \tl_clear_new:N \l_@@_borders_tikz_tl
6405         \keys_set:nV
6406             { NiceMatrix / OnlyForTikzInBorders }
6407         \l_@@_borders_clist
6408         \@@_cut_on_hyphen:w #2 \q_stop
6409         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6410         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6411         \@@_cut_on_hyphen:w #3 \q_stop
6412         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6413         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6414         \@@_stroke_borders_block_i:
6415     }
6416 }

6417 \hook_gput_code:nnn { begindocument } { . }
6418 {
6419     \cs_new_protected:Npx \@@_stroke_borders_block_i:
6420     {
6421         \c_@@_pgfortikzpicture_tl
6422         \@@_stroke_borders_block_ii:
6423         \c_@@_endpgfortikzpicture_tl
6424     }
6425 }

6426 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
6427 {
6428     \pgfrememberpicturepositiononpagetrue
6429     \pgf@relevantforpicturesizefalse
6430     \CT@arc@
6431     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
6432     \clist_if_in:NnT \l_@@_borders_clist { right }
6433     { \@@_stroke_vertical:n \l_tmpb_tl }
6434     \clist_if_in:NnT \l_@@_borders_clist { left }
6435     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
6436     \clist_if_in:NnT \l_@@_borders_clist { bottom }
6437     { \@@_stroke_horizontal:n \l_tmpa_tl }
6438     \clist_if_in:NnT \l_@@_borders_clist { top }
6439     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
6440 }

6441 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
6442 {
6443     tikz .code:n =
6444         \cs_if_exist:NTF \tikzpicture
6445         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
6446         { \@@_error:n { tikz-in~borders~without~tikz } } ,

```

```

6447   tikz .value_required:n = true ,
6448   top .code:n = ,
6449   bottom .code:n = ,
6450   left .code:n = ,
6451   right .code:n = ,
6452   unknown .code:n = \@@_error:n { bad~border }
6453 }

```

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).

```

6454 \cs_new_protected:Npn \@@_stroke_vertical:n #1
6455 {
6456   \@@_qpoint:n \l_@@_tmpc_tl
6457   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6458   \@@_qpoint:n \l_tmpa_tl
6459   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6460   \@@_qpoint:n { #1 }
6461   \tl_if_empty:NTF \l_@@_borders_tikz_tl
6462   {
6463     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
6464     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
6465     \pgfusepathqstroke
6466   }
6467   {
6468     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
6469     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
6470   }
6471 }

```

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).

```

6472 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
6473 {
6474   \@@_qpoint:n \l_@@_tmpd_tl
6475   \clist_if_in:NnTF \l_@@_borders_clist { left }
6476   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
6477   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
6478   \@@_qpoint:n \l_tmpb_tl
6479   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
6480   \@@_qpoint:n { #1 }
6481   \tl_if_empty:NTF \l_@@_borders_tikz_tl
6482   {
6483     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
6484     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6485     \pgfusepathqstroke
6486   }
6487   {
6488     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
6489     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
6490   }
6491 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

6492 \keys_define:nn { NiceMatrix / BlockBorders }
6493 {
6494   borders .clist_set:N = \l_@@_borders_clist ,
6495   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6496   rounded-corners .default:n = 4 pt ,
6497   line-width .dim_set:N = \l_@@_line_width_dim ,
6498 }

```

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.

```

6499 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
6500 {
6501   \begin { tikzpicture }
6502   \clist_map_inline:nn { #5 }
6503     {
6504       \path [ ##1 ]
6505         ( #1 -| #2 )
6506         rectangle
6507         ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
6508     }
6509   \end { tikzpicture }
6510 }

```

How to draw the dotted lines transparently

```

6511 \cs_set_protected:Npn \@@_renew_matrix:
6512 {
6513   \RenewDocumentEnvironment { pmatrix } { } {
6514     { \pNiceMatrix }
6515     { \endpNiceMatrix }
6516   \RenewDocumentEnvironment { vmatrix } { } {
6517     { \vNiceMatrix }
6518     { \endvNiceMatrix }
6519   \RenewDocumentEnvironment { Vmatrix } { } {
6520     { \VNiceMatrix }
6521     { \endVNiceMatrix }
6522   \RenewDocumentEnvironment { bmatrix } { } {
6523     { \bNiceMatrix }
6524     { \endbNiceMatrix }
6525   \RenewDocumentEnvironment { Bmatrix } { } {
6526     { \BNiceMatrix }
6527     { \endBNiceMatrix }
6528 }

```

Automatic arrays

```

6529 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
6530 {
6531   \int_set:Nn \l_@@_nb_rows_int { #1 }
6532   \int_set:Nn \l_@@_nb_cols_int { #2 }
6533 }

```

We will extract the potential keys `l`, `r` and `c` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

6534 \keys_define:nn { NiceMatrix / Auto }
6535 {
6536   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
6537   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
6538   c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
6539 }
6540 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
6541 {
6542   \int_zero_new:N \l_@@_nb_rows_int
6543   \int_zero_new:N \l_@@_nb_cols_int
6544   \@@_set_size:n #4 \q_stop

```

The group is for the protection of `\l_@@_type_of_col_tl`.

```

6545   \group_begin:

```

```

6546 \tl_set:Nn \l_@@_type_of_col_tl c
6547 \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
6548 \use:x
6549 {
6550   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
6551   { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
6552   [ \exp_not:V \l_tmpa_tl ]
6553 }
6554 \int_compare:nNnT \l_@@_first_row_int = 0
6555 {
6556   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6557   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6558   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6559 }
6560 \prg_replicate:nn \l_@@_nb_rows_int
6561 {
6562   \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).

```

6563   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
6564   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6565 }
6566 \int_compare:nNnT \l_@@_last_row_int > { -2 }
6567 {
6568   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6569   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6570   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6571 }
6572 \end { NiceArrayWithDelims }
6573 \group_end:
6574 }
6575 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6576 {
6577   \cs_set_protected:cpn { #1 AutoNiceMatrix }
6578   {
6579     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6580     \AutoNiceMatrixWithDelims { #2 } { #3 }
6581   }
6582 }
6583 \@@_define_com:nnn p ( )
6584 \@@_define_com:nnn b [ ]
6585 \@@_define_com:nnn v | |
6586 \@@_define_com:nnn V \l \l
6587 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

6588 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
6589 {
6590   \group_begin:
6591   \bool_set_true:N \l_@@_NiceArray_bool
6592   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6593   \group_end:
6594 }

```

The redefinition of the command \dotfill

```

6595 \cs_set_eq:NN \@@_old_dotfill \dotfill
6596 \cs_new_protected:Npn \@@_dotfill:
6597 {

```


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}`).

```

6598 \@@_old_dotfill
6599 \bool_if:NT \l_@@_NiceTabular_bool
6600 { \group_insert_after:N \@@_dotfill_ii: }
6601 { \group_insert_after:N \@@_dotfill_i: }
6602 }
6603 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
6604 \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`.

```

6605 \cs_new_protected:Npn \@@_dotfill_iii:
6606 { \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.

```

6607 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6608 {
6609   \tl_gput_right:Nx \g_@@_internal_code_after_tl
6610   {
6611     \@@_actually_diagbox:nnnnnn
6612     { \int_use:N \c@iRow }
6613     { \int_use:N \c@jCol }
6614     { \int_use:N \c@iRow }
6615     { \int_use:N \c@jCol }
6616     { \exp_not:n { #1 } }
6617     { \exp_not:n { #2 } }
6618   }

```

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`.

```

6619 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6620 {
6621   { \int_use:N \c@iRow }
6622   { \int_use:N \c@jCol }
6623   { \int_use:N \c@iRow }
6624   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

6625   { }
6626 }
6627 }

```

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.

```

6628 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
6629 {
6630   \pgfpicture
6631   \pgf@relevantforpicturesizefalse
6632   \pgfrememberpicturepositiononpagetrue
6633   \@@_qpoint:n { row - #1 }
6634   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6635   \@@_qpoint:n { col - #2 }
6636   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6637   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6638   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
6639   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y

```

```

6640 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
6641 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6642 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6643 {

```

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.

```

6644 \CT@arc@
6645 \pgfsetroundcap
6646 \pgfusepathqstroke
6647 }
6648 \pgfset { inner~sep = 1 pt }
6649 \pgfscope
6650 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6651 \pgfnode { rectangle } { south-west }
6652 {
6653 \begin { minipage } { 20 cm }
6654 \@@_math_toggle_token: #5 \@@_math_toggle_token:
6655 \end { minipage }
6656 }
6657 { }
6658 { }
6659 \endpgfscope
6660 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6661 \pgfnode { rectangle } { north-east }
6662 {
6663 \begin { minipage } { 20 cm }
6664 \raggedleft
6665 \@@_math_toggle_token: #6 \@@_math_toggle_token:
6666 \end { minipage }
6667 }
6668 { }
6669 { }
6670 \endpgfpicture
6671 }

```

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.

```

6672 \keys_define:nn { NiceMatrix }
6673 {
6674 CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6675 CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6676 }
6677 \keys_define:nn { NiceMatrix / CodeAfter }
6678 {
6679 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6680 sub-matrix .value_required:n = true ,
6681 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6682 delimiters / color .value_required:n = true ,
6683 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6684 rules .value_required:n = true ,
6685 unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
6686 }

```

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. 122.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
6687 \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 `\`.

```
6688 \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`.

```
6689 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
6690 {
6691   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6692   \@@_CodeAfter_iv:n
6693 }
```

We catch the argument of the command `\end` (in `#1`).

```
6694 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
6695 {
```

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.

```
6696   \str_if_eq:eeTF \@currenvir { #1 } { \end { #1 } }
```

If this is not the `\end` we are looking for, we put those tokens in `\g_nicematrix_code_after_tl` and we go on searching for the next command `\end` with a recursive call to the command `\@@_CodeAfter:n`.

```
6697   {
6698     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6699     \@@_CodeAfter_ii:n
6700   }
6701 }
```

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.

```
6702 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
6703 {
6704   \pgfpicture
6705   \pgfrememberpicturepositiononpagetrue
6706   \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.

```
6707   \@@_qpoint:n { row - 1 }
6708   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6709   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
6710   \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).

```
6711   \bool_if:nTF { #3 }
6712     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6713     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6714   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
```

```

6715 {
6716   \cs_if_exist:cT
6717   { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
6718   {
6719     \pgfpointanchor
6720     { \l_@@_env: - ##1 - #2 }
6721     { \bool_if:nTF { #3 } { west } { east } }
6722     \dim_set:Nn \l_tmpa_dim
6723     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6724   }
6725 }

```

Now we can put the delimiter with a node of PGF.

```

6726 \pgfset { inner~sep = \c_zero_dim }
6727 \dim_zero:N \nulldelimiterspace
6728 \pgftransformshift
6729 {
6730   \pgfpoint
6731   { \l_tmpa_dim }
6732   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6733 }
6734 \pgfnode
6735 { rectangle }
6736 { \bool_if:nTF { #3 } { east } { west } }
6737 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

6738   \nullfont
6739   \c_math_toggle_token
6740   \tl_if_empty:NF \l_@@_delimiters_color_tl
6741   { \color { \l_@@_delimiters_color_tl } }
6742   \bool_if:nTF { #3 } { \left #1 } { \left . }
6743   \vcenter
6744   {
6745     \nullfont
6746     \hrule \@height
6747       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6748       \@depth \c_zero_dim
6749       \@width \c_zero_dim
6750   }
6751   \bool_if:nTF { #3 } { \right . } { \right #1 }
6752   \c_math_toggle_token
6753 }
6754 { }
6755 { }
6756 \endpgfpicture
6757 }

```

The command `\SubMatrix`

```

6758 \keys_define:nn { NiceMatrix / sub-matrix }
6759 {
6760   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6761   extra-height .value_required:n = true ,
6762   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6763   left-xshift .value_required:n = true ,
6764   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6765   right-xshift .value_required:n = true ,
6766   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6767   xshift .value_required:n = true ,
6768   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6769   delimiters / color .value_required:n = true ,
6770   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6771   slim .default:n = true ,

```

```

6772 hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6773 hlines .default:n = all ,
6774 vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6775 vlines .default:n = all ,
6776 hvlines .meta:n = { hlines, vlines } ,
6777 hvlines .value_forbidden:n = true ,
6778 }
6779 \keys_define:nn { NiceMatrix }
6780 {
6781   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6782   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6783   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6784   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6785   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6786   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6787 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

6788 \keys_define:nn { NiceMatrix / SubMatrix }
6789 {
6790   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6791   delimiters / color .value_required:n = true ,
6792   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6793   hlines .default:n = all ,
6794   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6795   vlines .default:n = all ,
6796   hvlines .meta:n = { hlines, vlines } ,
6797   hvlines .value_forbidden:n = true ,
6798   name .code:n =
6799     \tl_if_empty:nTF { #1 }
6800     { \@@_error:n { Invalid-name-format } }
6801     {
6802       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6803       {
6804         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6805         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
6806         {
6807           \str_set:Nn \l_@@_submatrix_name_str { #1 }
6808           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6809         }
6810       }
6811       { \@@_error:n { Invalid-name-format } }
6812     } ,
6813   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6814   rules .value_required:n = true ,
6815   code .tl_set:N = \l_@@_code_tl ,
6816   code .value_required:n = true ,
6817   name .value_required:n = true ,
6818   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
6819 }

6820 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
6821 {
6822   \peek_remove_spaces:n
6823   {
6824     \@@_cut_on_hyphen:w #3 \q_stop
6825     \tl_clear_new:N \l_@@_tmpc_tl
6826     \tl_clear_new:N \l_@@_tmpd_tl
6827     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6828     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6829     \@@_cut_on_hyphen:w #2 \q_stop
6830     \seq_gput_right:Nx \g_@@_submatrix_seq

```

```

6831      { { \l_tmpa_tl } { \l_tmpb_tl } { \l_@@_tmpc_tl } { \l_@@_tmpd_tl } }
6832      \tl_gput_right:Nn \g_@@_internal_code_after_tl
6833      { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6834    }
6835  }

```

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.

```

6836 \hook_gput_code:nnn { begindocument } { . }
6837 {
6838   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
6839   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
6840   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
6841     {
6842       \peek_remove_spaces:n
6843       {
6844         \@@_sub_matrix:nnnnnnn
6845         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
6846       }
6847     }
6848 }

```

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).

```

6849 \cs_new_protected:Npn \@@_compute_i_j:nn #1 #2
6850 {
6851   \tl_clear_new:N \l_@@_first_i_tl
6852   \tl_clear_new:N \l_@@_first_j_tl
6853   \tl_clear_new:N \l_@@_last_i_tl
6854   \tl_clear_new:N \l_@@_last_j_tl
6855   \@@_cut_on_hyphen:w #1 \q_stop
6856   \tl_if_eq:NnTF \l_tmpa_tl { last }
6857     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
6858     { \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl }
6859   \tl_if_eq:NnTF \l_tmpb_tl { last }
6860     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
6861     { \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl }
6862   \@@_cut_on_hyphen:w #2 \q_stop
6863   \tl_if_eq:NnTF \l_tmpa_tl { last }
6864     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
6865     { \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl }
6866   \tl_if_eq:NnTF \l_tmpb_tl { last }
6867     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
6868     { \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl }
6869 }

```

```

6870 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6871 {
6872   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

6873   \@@_compute_i_j:nn { #2 } { #3 }
6874   \bool_lazy_or:nnTF
6875     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6876     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6877     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
6878     {
6879       \str_clear_new:N \l_@@_submatrix_name_str
6880       \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6881       \pgfpicture
6882       \pgfrememberpicturepositiononpagetrue
6883       \pgf@relevantforpicturesizefalse
6884       \pgfset { inner~sep = \c_zero_dim }
6885       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6886       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currification.

```

6887       \bool_if:NTF \l_@@_submatrix_slim_bool
6888         { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6889         { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6890         {
6891           \cs_if_exist:cT
6892             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6893             {
6894               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6895               \dim_set:Nn \l_@@_x_initial_dim
6896                 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6897             }
6898           \cs_if_exist:cT
6899             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6900             {
6901               \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6902               \dim_set:Nn \l_@@_x_final_dim
6903                 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6904             }
6905         }
6906       \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
6907         { \@@_error:nn { impossible-delimiter } { left } }
6908         {
6909           \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
6910             { \@@_error:nn { impossible-delimiter } { right } }
6911             { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
6912         }
6913       \endpgfpicture
6914     }
6915   \group_end:
6916 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

6917 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
6918 {
6919   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
6920   \dim_set:Nn \l_@@_y_initial_dim
6921     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
6922   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
6923   \dim_set:Nn \l_@@_y_final_dim
6924     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
6925   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
6926     {
6927       \cs_if_exist:cT

```

```

6928     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
6929     {
6930         \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
6931         \dim_set:Nn \l_@@_y_initial_dim
6932             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6933     }
6934     \cs_if_exist:cT
6935     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6936     {
6937         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6938         \dim_set:Nn \l_@@_y_final_dim
6939             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6940     }
6941 }
6942 \dim_set:Nn \l_tmpa_dim
6943 {
6944     \l_@@_y_initial_dim - \l_@@_y_final_dim +
6945     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6946 }
6947 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

6948     \group_begin:
6949     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6950     \tl_if_empty:NF \l_@@_rules_color_tl
6951     { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6952     \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`.

```

6953     \seq_map_inline:Nn \g_@@_cols_vlism_seq
6954     {
6955         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6956         {
6957             \int_compare:nNnT
6958                 { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6959             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

6960                 \@@_qpoint:n { col - ##1 }
6961                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6962                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6963                 \pgfusepathqstroke
6964             }
6965         }
6966     }

```

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.

```

6967     \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6968     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6969     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6970     {
6971         \bool_lazy_and:nnTF
6972             { \int_compare_p:nNn { ##1 } > 0 }
6973             {
6974                 \int_compare_p:nNn
6975                     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6976             {
6977                 \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6978                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6979                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }

```



```

6980         \pgfusepathqstroke
6981     }
6982     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
6983 }

```

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.

```

6984     \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6985     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6986     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
6987     {
6988         \bool_lazy_and:nnTF
6989         { \int_compare_p:nNn { ##1 } > 0 }
6990         {
6991             \int_compare_p:nNn
6992             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6993         {
6994             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

6995         \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
6996         \dim_set:Nn \l_tmpa_dim
6997         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6998         \str_case:nn { #1 }
6999         {
7000             ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7001             [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7002             \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7003         }
7004         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

7005         \dim_set:Nn \l_tmpb_dim
7006         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7007         \str_case:nn { #2 }
7008         {
7009             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7010             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7011             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7012         }
7013         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7014         \pgfusepathqstroke
7015         \group_end:
7016     }
7017     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7018 }

```

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).

```

7019     \str_if_empty:NF \l_@@_submatrix_name_str
7020     {
7021         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
7022         \l_@@_x_initial_dim \l_@@_y_initial_dim
7023         \l_@@_x_final_dim \l_@@_y_final_dim
7024     }
7025     \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`.

```

7026     \begin { pgfscope }
7027     \pgftransformshift

```

```

7028 {
7029   \pgfpoint
7030   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7031   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7032 }
7033 \str_if_empty:NTF \l_@@_submatrix_name_str
7034 { \@@_node_left:nn #1 { } }
7035 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7036 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7037 \pgftransformshift
7038 {
7039   \pgfpoint
7040   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7041   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7042 }
7043 \str_if_empty:NTF \l_@@_submatrix_name_str
7044 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7045 {
7046   \@@_node_right:nnnn #2
7047   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7048 }
7049 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7050 \flag_clear_new:n { nicematrix }
7051 \l_@@_code_tl
7052 }

```

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`.

```

7053 \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.

```

7054 \cs_new_protected:Npn \@@_pgfpntanchor:n #1
7055 {
7056   \use:e
7057   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpntanchor_i:nn #1 } }
7058 }

```

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 -.

```

7059 \cs_new:Npn \@@_pgfpntanchor_i:nn #1 #2
7060 { #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.

```

7061 \tl_const:Nn \c_@@_integers_alist_tl
7062 {
7063   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7064   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7065   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7066   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7067 }

```

```

7068 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
7069 {

```

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`.

```

7070   \tl_if_empty:nTF { #2 }
7071   {
7072     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7073     {
7074       \flag_raise:n { nicematrix }
7075       \int_if_even:nTF { \flag_height:n { nicematrix } }
7076       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7077       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7078     }
7079     { #1 }
7080   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row}-i$ or $\text{col}-j$.

```

7081   { \@@_pgfpointanchor_iii:w { #1 } #2 }
7082 }

```

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`).

```

7083 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
7084 {
7085   \str_case:nnF { #1 }
7086   {
7087     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7088     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7089   }

```

Now the case of a node of the form $i-j$.

```

7090   {
7091     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7092     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7093   }
7094 }

```

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`).

```

7095 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7096 {
7097   \pgfnode
7098   { rectangle }
7099   { east }
7100   {
7101     \nullfont
7102     \c_math_toggle_token
7103     \tl_if_empty:NF \l_@@_delimiters_color_tl
7104     { \color { \l_@@_delimiters_color_tl } }
7105     \left #1
7106     \vcenter
7107     {
7108       \nullfont
7109       \hrule \@height \l_tmpa_dim
7110       \@depth \c_zero_dim
7111       \@width \c_zero_dim
7112     }

```

```

7113     \right .
7114     \c_math_toggle_token
7115   }
7116   { #2 }
7117   { }
7118 }

```

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.

```

7119 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
7120 {
7121   \pgfnode
7122   { rectangle }
7123   { west }
7124   {
7125     \nullfont
7126     \c_math_toggle_token
7127     \tl_if_empty:NF \l_@@_delimiters_color_tl
7128     { \color { \l_@@_delimiters_color_tl } }
7129     \left .
7130     \vcenter
7131     {
7132       \nullfont
7133       \hrule \@height \l_tmpa_dim
7134             \@depth \c_zero_dim
7135             \@width \c_zero_dim
7136     }
7137     \right #1
7138     \tl_if_empty:NF { #3 } { _ { \smash { #3 } } }
7139     ^ { \smash { #4 } }
7140     \c_math_toggle_token
7141   }
7142   { #2 }
7143   { }
7144 }

```

Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

7145 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
7146 {
7147   \peek_remove_spaces:n
7148   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
7149 }
7150 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
7151 {
7152   \peek_remove_spaces:n
7153   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
7154 }
7155 \keys_define:nn { NiceMatrix / Brace }
7156 {
7157   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
7158   left-shorten .default:n = true ,
7159   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
7160   shorten .meta:n = { left-shorten , right-shorten } ,
7161   right-shorten .default:n = true ,
7162   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
7163   yshift .value_required:n = true ,

```

```

7164 yshift .initial:n = \c_zero_dim ,
7165 color .tl_set:N = \l_tmpa_tl ,
7166 color .value_required:n = true ,
7167 unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
7168 }

```

#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.

```

7169 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
7170 {
7171   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

7172   \@@_compute_i_j:nn { #1 } { #2 }
7173   \bool_lazy_or:nnTF
7174     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7175     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7176     {
7177       \str_if_eq:nnTF { #5 } { under }
7178       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
7179       { \@@_error:nn { Construct~too~large } { \OverBrace } }
7180     }
7181   {
7182     \tl_clear:N \l_tmpa_tl % added the 2022-02-25
7183     \keys_set:nn { NiceMatrix / Brace } { #4 }
7184     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } } % added the 2022-02-25
7185     \pgfpicture
7186     \pgfrememberpicturepositiononpagetrue
7187     \pgf@relevantforpicturesizefalse
7188     \bool_if:NT \l_@@_brace_left_shorten_bool
7189     {
7190       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7191       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7192       {
7193         \cs_if_exist:cT
7194           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7195           {
7196             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7197             \dim_set:Nn \l_@@_x_initial_dim
7198               { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7199           }
7200       }
7201     }
7202     \bool_lazy_or:nnT
7203       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
7204       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
7205       {
7206         \@@_qpoint:n { col - \l_@@_first_j_tl }
7207         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7208       }
7209     \bool_if:NT \l_@@_brace_right_shorten_bool
7210     {
7211       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
7212       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7213       {
7214         \cs_if_exist:cT
7215           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7216           {
7217             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7218             \dim_set:Nn \l_@@_x_final_dim
7219               { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7220           }
7221       }

```

```

7222     }
7223     \bool_lazy_or:nnT
7224     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
7225     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
7226     {
7227         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
7228         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7229     }
7230     \pgfset { inner-sep = \c_zero_dim }
7231     \str_if_eq:nnTF { #5 } { under }
7232     { \@@_underbrace_i:n { #3 } }
7233     { \@@_overbrace_i:n { #3 } }
7234     \endpgfpicture
7235 }
7236 \group_end:
7237 }

```

The argument is the text to put above the brace.

```

7238 \cs_new_protected:Npn \@@_overbrace_i:n #1
7239 {
7240     \@@_qpoint:n { row - \l_@@_first_i_tl }
7241     \pgftransformshift
7242     {
7243         \pgfpoint
7244         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7245         { \pgf@y + \l_@@_brace_yshift_dim }
7246     }
7247     \pgfnode
7248     { rectangle }
7249     { south }
7250     {
7251         \vbox_top:n
7252         {
7253             \group_begin:
7254             \everycr { }
7255             \halign
7256             {
7257                 \hfil ## \hfil \crrc
7258                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7259                 \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7260                 \hbox_to_wd:nn
7261                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7262                 { \downbracefill } \cr
7263             }
7264             \group_end:
7265         }
7266     }
7267     { }
7268     { }
7269 }

```

The argument is the text to put under the brace.

```

7270 \cs_new_protected:Npn \@@_underbrace_i:n #1
7271 {
7272     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
7273     \pgftransformshift
7274     {
7275         \pgfpoint
7276         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7277         { \pgf@y - \l_@@_brace_yshift_dim }
7278     }
7279     \pgfnode
7280     { rectangle }

```

```

7281 { north }
7282 {
7283   \group_begin:
7284   \everycr { }
7285   \vbox:n
7286   {
7287     \halign
7288     {
7289       \hfil ## \hfil \crcr
7290       \hbox_to_wd:nn
7291       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7292       { \upbracefill } \cr
7293       \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7294       \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7295     }
7296   }
7297   \group_end:
7298 }
7299 { }
7300 { }
7301 }

```

We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

7302 \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.

```

7303 \bool_new:N \c_@@_footnote_bool

7304 \@@_msg_new:nnn { Unknown~key~for~package }
7305 {
7306   The~key~'\l_keys_key_str'~is~unknown. \\\
7307   If~you~go~on,~it~will~be~ignored. \\\
7308   For~a~list~of~the~available~keys,~type~H~<return>.
7309 }
7310 {
7311   The~available~keys~are~(in~alphabetic~order):~
7312   footnote,~
7313   footnotehyper,~
7314   renew-dots,~and
7315   renew-matrix.
7316 }

7317 \keys_define:nn { NiceMatrix / Package }
7318 {
7319   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
7320   renew-dots .value_forbidden:n = true ,
7321   renew-matrix .code:n = \@@_renew_matrix: ,
7322   renew-matrix .value_forbidden:n = true ,
7323   transparent .code:n = \@@_fatal:n { Key~transparent } ,
7324   transparent .value_forbidden:n = true,
7325   footnote .bool_set:N = \c_@@_footnote_bool ,
7326   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
7327   unknown .code:n = \@@_error:n { Unknown~key~for~package }

```

```

7328 }
7329 \ProcessKeysOptions { NiceMatrix / Package }

7330 \@@_msg_new:nn { footnote~with~footnotehyper~package }
7331 {
7332   You~can't~use~the~option~'footnote'~because~the~package~
7333   footnotehyper~has~already~been~loaded.~
7334   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
7335   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7336   of~the~package~footnotehyper.\\
7337   If~you~go~on,~the~package~footnote~won't~be~loaded.
7338 }

7339 \@@_msg_new:nn { footnotehyper~with~footnote~package }
7340 {
7341   You~can't~use~the~option~'footnotehyper'~because~the~package~
7342   footnote~has~already~been~loaded.~
7343   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
7344   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7345   of~the~package~footnote.\\
7346   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
7347 }

7348 \bool_if:NT \c_@@_footnote_bool
7349 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7350   \@ifclassloaded { beamer }
7351   { \bool_set_false:N \c_@@_footnote_bool }
7352   {
7353     \@ifpackageloaded { footnotehyper }
7354     { \@_error:n { footnote~with~footnotehyper~package } }
7355     { \usepackage { footnote } }
7356   }
7357 }

7358 \bool_if:NT \c_@@_footnotehyper_bool
7359 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7360   \@ifclassloaded { beamer }
7361   { \bool_set_false:N \c_@@_footnote_bool }
7362   {
7363     \@ifpackageloaded { footnote }
7364     { \@_error:n { footnotehyper~with~footnote~package } }
7365     { \usepackage { footnotehyper } }
7366   }
7367   \bool_set_true:N \c_@@_footnote_bool
7368 }

```

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

```

7369 \seq_new:N \c_@@_types_of_matrix_seq
7370 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
7371 {
7372   NiceMatrix ,
7373   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix

```



```

7374 }
7375 \seq_set_map_x:Nn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
7376 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is executed. This command raises an error but try to give the best information to the user in the error message. The command `\seq_if_in:NVTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

7377 \cs_new_protected:Npn \@@_error_too_much_cols:
7378 {
7379   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
7380   {
7381     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
7382     { \@@_fatal:n { too-much-cols-for-matrix } }
7383     {
7384       \bool_if:NF \l_@@_last_col_without_value_bool
7385       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
7386     }
7387   }
7388   { \@@_fatal:n { too-much-cols-for-array } }
7389 }

```

The following command must *not* be protected since it's used in an error message.

```

7390 \cs_new:Npn \@@_message_hdotsfor:
7391 {
7392   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
7393   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is~incorrect.}
7394 }
7395 \@@_msg_new:nn { negative-weight }
7396 {
7397   The-weight-of-the~'X'~columns-must-be-positive-and-you-have-used~
7398   the-value~'#1'.~If-you-go-on,~the-absolute-value-will-be-used.
7399 }
7400 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
7401 {
7402   You-try-to-use-more-columns-than-allowed-by-your~
7403   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of~
7404   columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus-the~
7405   exterior-columns).~This-error-is-fatal.
7406 }
7407 \@@_msg_new:nn { too-much-cols-for-matrix }
7408 {
7409   You-try-to-use-more-columns-than-allowed-by-your~
7410   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
7411   number-of-columns-for-a-matrix-is-fixed-by-the-LaTeX-counter~
7412   'MaxMatrixCols'.~Its-actual-value-is~\int_use:N \c@MaxMatrixCols.~
7413   This-error-is-fatal.
7414 }

```

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`.

```

7415 \@@_msg_new:nn { too-much-cols-for-array }
7416 {
7417   You-try-to-use-more-columns-than-allowed-by-your~
7418   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
7419   \int_use:N \g_@@_static_num_of_col_int\
7420   ~(plus-the-potential-exterior-ones).~
7421   This-error-is-fatal.
7422 }
7423 \@@_msg_new:nn { hvlines-except-corners }
7424 {
7425   The-key~'hvlines-except-corners'~is-now-obsolete.~You-should~instead-use-the~

```

```

7426 keys~'hvlines'~and~'corners'.\\
7427 However,~you~can~go~on~for~this~time.~This~message~won't~be~shown~anymore~
7428 in~this~document.
7429 }
7430 \\@@_msg_new:nn { last~col~not~used }
7431 {
7432 The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
7433 in~your~\\@@_full_name_env:.~However,~you~can~go~on.
7434 }
7435 \\@@_msg_new:nn { columns~not~used }
7436 {
7437 The~preamble~of~your~\\@@_full_name_env:\\ announces~\\int_use:N
7438 \\g_@@_static_num_of_col_int\\ columns~but~you~use~only~\\int_use:N \\c@jCol.\\
7439 You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
7440 }
7441 \\@@_msg_new:nn { in~first~col }
7442 {
7443 You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
7444 If~you~go~on,~this~command~will~be~ignored.
7445 }
7446 \\@@_msg_new:nn { in~last~col }
7447 {
7448 You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
7449 If~you~go~on,~this~command~will~be~ignored.
7450 }
7451 \\@@_msg_new:nn { in~first~row }
7452 {
7453 You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
7454 If~you~go~on,~this~command~will~be~ignored.
7455 }
7456 \\@@_msg_new:nn { in~last~row }
7457 {
7458 You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
7459 If~you~go~on,~this~command~will~be~ignored.
7460 }
7461 \\@@_msg_new:nn { double~closing~delimiter }
7462 {
7463 You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
7464 delimiter.~This~delimiter~will~be~ignored.
7465 }
7466 \\@@_msg_new:nn { delimiter~after~opening }
7467 {
7468 You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
7469 delimiter.~This~delimiter~will~be~ignored.
7470 }
7471 \\@@_msg_new:nn { bad~option~for~line~style }
7472 {
7473 Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
7474 is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
7475 }
7476 \\@@_msg_new:nn { Unknown~key~for~xdots }
7477 {
7478 As~for~now,~there~is~only~three~keys~available~here:~'color',~'line~style'~
7479 and~'shorten'~(and~you~try~to~use~'\\l_keys_key_str').~If~you~go~on,~
7480 this~key~will~be~ignored.
7481 }
7482 \\@@_msg_new:nn { Unknown~key~for~rowcolors }
7483 {
7484 As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~

```

```

7485     (and-you-try-to-use-'l_keys_key_str').~If-you-go-on,~
7486     this-key-will-be-ignored.
7487 }

7488 \@@_msg_new:nn { ampersand-in-light-syntax }
7489 {
7490     You-can't-use-an-ampersand-(\token_to_str:N &)~to-separate-columns-because~
7491     ~you-have-used-the-key~'light-syntax'.~This-error-is-fatal.
7492 }

7493 \@@_msg_new:nn { Construct-too-large }
7494 {
7495     Your-command-\token_to_str:N #1
7496     can't-be-drawn-because-your-matrix-is-too-small.\\
7497     If-you-go-on,~this-command-will-be-ignored.
7498 }

7499 \@@_msg_new:nn { double-backslash-in-light-syntax }
7500 {
7501     You-can't-use-\token_to_str:N \\~to-separate-rows-because-you-have-used~
7502     the-key~'light-syntax'.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
7503     (set-by-the-key~'end-of-row').~This-error-is-fatal.
7504 }

7505 \@@_msg_new:nn { standard-cline-in-document }
7506 {
7507     The-key~'standard-cline'~is-available-only-in-the-preamble.\\
7508     If-you-go-on-this-command-will-be-ignored.
7509 }

7510 \@@_msg_new:nn { bad-value-for-baseline }
7511 {
7512     The-value-given-to~'baseline'~(\int_use:N \l_tmpa_int)~is-not~
7513     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
7514     \int_use:N \g_@@_row_total_int\ or-equal-to~'t',~'c'~or~'b'.\\
7515     If-you-go-on,~a-value-of-1-will-be-used.
7516 }

7517 \@@_msg_new:nn { Invalid-name-format }
7518 {
7519     You-can't-give-the-name~'\l_keys_value_tl'~to-a-\token_to_str:N
7520     \SubMatrix.\\
7521     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
7522     If-you-go-on,~this-key-will-be-ignored.
7523 }

7524 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
7525 {
7526     You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
7527     \token_to_str:N \SubMatrix\ of-your~\@@_full_name_env:\ but~that~
7528     number-is-not-valid.~If-you-go-on,~it-will-be-ignored.
7529 }

7530 \@@_msg_new:nn { impossible-delimiter }
7531 {
7532     It's-impossible-to-draw-the~#1~delimiter~of-your~
7533     \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
7534     in-that-column.
7535     \bool_if:NT \l_@@_submatrix_slim_bool
7536     { ~Maybe-you-should-try-without-the-key~'slim'. } \\
7537     If-you-go-on,~this~\token_to_str:N \SubMatrix\ will-be-ignored.
7538 }

7539 \@@_msg_new:nn { width-without-X-columns }
7540 {
7541     You-have-used-the-key~'width'~but-you-have-put-no~'X'~column. \\
7542     If-you-go-on,~that-key-will-be-ignored.
7543 }

```

```

7544 \@@_msg_new:nn { empty-environment }
7545 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }

7546 \@@_msg_new:nn { Wrong-use-of~v-center }
7547 {
7548   You~should~not~use~the~key~'v-center'~here~because~your~block~is~not~
7549   mono~row.~However,~you~can~go~on.
7550 }

7551 \@@_msg_new:nn { No~letter~and~no~command }
7552 {
7553   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
7554   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~key~'command'~
7555   (to~draw~horizontal~rules).\\
7556   However,~you~can~go~on.
7557 }

7558 \@@_msg_new:nn { Forbidden~letter }
7559 {
7560   You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
7561   If~you~go~on,~it~will~be~ignored.
7562 }

7563 \@@_msg_new:nn { key~width~without~key~tikz }
7564 {
7565   In~'custom-line',~you~have~used~'width'~without~'tikz'.~That's~not~correct.~
7566   If~you~go~on,~that~key~'width'~will~be~discarded.
7567 }

7568 \@@_msg_new:nn { Several~letters }
7569 {
7570   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~
7571   have~used~'\l_@@_letter_str').\\
7572   If~you~go~on,~it~will~be~ignored.
7573 }

7574 \@@_msg_new:nn { Delimiter~with~small }
7575 {
7576   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
7577   because~the~key~'small'~is~in~force.\\
7578   This~error~is~fatal.
7579 }

7580 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
7581 {
7582   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code~after'~
7583   can't~be~executed~because~a~cell~doesn't~exist.\\
7584   If~you~go~on~this~command~will~be~ignored.
7585 }

7586 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
7587 {
7588   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
7589   in~this~\@@_full_name_env:.\
7590   If~you~go~on,~this~key~will~be~ignored.\\
7591   For~a~list~of~the~names~already~used,~type~H~<return>.
7592 }
7593 {
7594   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
7595   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
7596 }

7597 \@@_msg_new:nn { r~or~l~with~preamble }
7598 {
7599   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
7600   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
7601   your~\@@_full_name_env:.\
7602   If~you~go~on,~this~key~will~be~ignored.
7603 }

```

```

7604 \@@_msg_new:nn { Hdotsfor~in~col~0 }
7605 {
7606   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
7607   the~array.~This~error~is~fatal.
7608 }
7609 \@@_msg_new:nn { bad~corner }
7610 {
7611   #1~is~an~incorrect~specification~for~a~corner~(in~the~keys~
7612   'corners'~and~'except~corners').~The~available~
7613   values~are:~NW,~SW,~NE~and~SE.\\
7614   If~you~go~on,~this~specification~of~corner~will~be~ignored.
7615 }
7616 \@@_msg_new:nn { bad~border }
7617 {
7618   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
7619   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
7620   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
7621   also~use~the~key~'tikz'
7622   \bool_if:nF \c_@@_tikz_loaded_bool
7623   {~if~you~load~the~LaTeX~package~'tikz'}).\\
7624   If~you~go~on,~this~specification~of~border~will~be~ignored.
7625 }
7626 \@@_msg_new:nn { tikz~key~without~tikz }
7627 {
7628   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
7629   \Block'~because~you~have~not~loaded~Tikz.~
7630   If~you~go~on,~this~key~will~be~ignored.
7631 }
7632 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
7633 {
7634   In~the~\@@_full_name_env:,~you~must~use~the~key~
7635   'last~col'~without~value.\\
7636   However,~you~can~go~on~for~this~time~
7637   (the~value~'\l_keys_value_tl'~will~be~ignored).
7638 }
7639 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
7640 {
7641   In~\NiceMatrixoptions,~you~must~use~the~key~
7642   'last~col'~without~value.\\
7643   However,~you~can~go~on~for~this~time~
7644   (the~value~'\l_keys_value_tl'~will~be~ignored).
7645 }
7646 \@@_msg_new:nn { Block~too~large~1 }
7647 {
7648   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
7649   too~small~for~that~block. \\
7650 }
7651 \@@_msg_new:nn { Block~too~large~2 }
7652 {
7653   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7654   \g_@@_static_num_of_col_int\
7655   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
7656   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
7657   (&)~at~the~end~of~the~first~row~of~your~
7658   \@@_full_name_env:.\
7659   If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
7660 }
7661 \@@_msg_new:nn { unknown~column~type }
7662 {
7663   The~column~type~'#1'~in~your~\@@_full_name_env:\

```

```

7664     is-unknown. \\
7665     This-error-is-fatal.
7666 }

7667 \@@_msg_new:nn { colon-without-arydshln }
7668 {
7669     The-column-type~': '~in-your~\@@_full_name_env~\
7670     is-unknown.~If-you-want-to-use~': '~of~'arydshln',~you-should~
7671     load-that-package.~If-you-want~a-dotted-line-of~'nicematrix',~you~
7672     should-use~'\l_@@_letter_for_dotted_lines_str'.\\
7673     This-error-is-fatal.
7674 }

7675 \@@_msg_new:nn { tabularnote-forbidden }
7676 {
7677     You-can't-use-the-command~\token_to_str:N\tabularnote\
7678     ~in-a~\@@_full_name_env:.~This-command-is-available-only-in-
7679     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
7680     If-you-go-on,~this-command-will-be-ignored.
7681 }

7682 \@@_msg_new:nn { borders-forbidden }
7683 {
7684     You-can't-use-the-key~'borders'~of-the-command~\token_to_str:N \Block\
7685     because-the-option~'rounded-corners'~
7686     is-in-force-with-a-non-zero-value.\\
7687     If-you-go-on,~this-key-will-be-ignored.
7688 }

7689 \@@_msg_new:nn { bottomrule-without-booktabs }
7690 {
7691     You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
7692     loaded~'booktabs'.\\
7693     If-you-go-on,~this-key-will-be-ignored.
7694 }

7695 \@@_msg_new:nn { enumitem-not-loaded }
7696 {
7697     You-can't-use-the-command~\token_to_str:N\tabularnote\
7698     ~because-you-haven't-loaded~'enumitem'.\\
7699     If-you-go-on,~this-command-will-be-ignored.
7700 }

7701 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
7702 {
7703     You-have-used-the-key~'tikz'~in-the-definition-of-a~
7704     customized-line~(with~'custom-line')~but-Tikz-is-not-loaded.~
7705     You-can-go-on-but-you-will-have-another-error-if-you-actually~
7706     use-that-custom-line.
7707 }

7708 \@@_msg_new:nn { tikz-in-borders-without-tikz }
7709 {
7710     You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
7711     command~'\token_to_str:N\Block')~but-Tikz-is-not-loaded.~
7712     If-you-go-on,~that-key-will-be-ignored.
7713 }

7714 \@@_msg_new:nn { color-in-custom-line-with-tikz }
7715 {
7716     In-a~'custom-line',~you-have-used-both~'tikz'~and~'color',~
7717     which-is-forbidden~(you-should-use~'color'~inside-the-key~'tikz').~
7718     If-you-go-on,~the-key~'color'~will-be-discarded.
7719 }

7720 \@@_msg_new:nn { Wrong-last-row }
7721 {
7722     You-have-used~'last-row=\int_use:N \l_@@_last_row_int'~but-your~
7723     \@@_full_name_env~ seems-to-have~\int_use:N \c@iRow \ rows.~

```

```

7724   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
7725   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
7726   without~value~(more~compilations~might~be~necessary).
7727 }

7728 \@@_msg_new:nn { Yet~in~env }
7729 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }

7730 \@@_msg_new:nn { Outside~math~mode }
7731 {
7732   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
7733   (and~not~in~\token_to_str:N \vcenter).\\
7734   This~error~is~fatal.
7735 }

7736 \@@_msg_new:nn { One~letter~allowed }
7737 {
7738   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
7739   If~you~go~on,~it~will~be~ignored.
7740 }

7741 \@@_msg_new:nn { varwidth~not~loaded }
7742 {
7743   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
7744   loaded.\\
7745   If~you~go~on,~your~column~will~behave~like~'p'.
7746 }

7747 \@@_msg_new:nnn { Unknown~key~for~Block }
7748 {
7749   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
7750   \Block.\\ If~you~go~on,~it~will~be~ignored. \\
7751   For~a~list~of~the~available~keys,~type~H<return>.
7752 }
7753 {
7754   The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
7755   hlines,~hvlines,~l,~line-width,~name,~rounded-corners,~r,~respect-arraystretch,
7756   ~t,~tikz~and~vlines.
7757 }

7758 \@@_msg_new:nn { Version~of~siunitx~too~old }
7759 {
7760   You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
7761   is~too~old.~You~need~at~least~v3.0.\\
7762   This~error~is~fatal.
7763 }

7764 \@@_msg_new:nnn { Unknown~key~for~Brace }
7765 {
7766   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
7767   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
7768   If~you~go~on,~it~will~be~ignored. \\
7769   For~a~list~of~the~available~keys,~type~H<return>.
7770 }
7771 {
7772   The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
7773   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
7774   right-shorten)~and~yshift.
7775 }

7776 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
7777 {
7778   The~key~'\l_keys_key_str'~is~unknown.\\
7779   If~you~go~on,~it~will~be~ignored. \\
7780   For~a~list~of~the~available~keys~in~\token_to_str:N
7781   \CodeAfter,~type~H<return>.
7782 }
7783 {

```

```

7784 The~available~keys~are~(in~alphabetic~order):~
7785 delimiters/color,~
7786 rules~(with~the~subkeys~'color'~and~'width'),~
7787 sub-matrix~(several~subkeys)~
7788 and~xdots~(several~subkeys).~
7789 The~latter~is~for~the~command~\token_to_str:N \line.
7790 }
7791 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
7792 {
7793   The~key~'\l_keys_key_str'~is~unknown.\\
7794   If~you~go~on,~this~key~will~be~ignored. \\
7795   For~a~list~of~the~available~keys~in~\token_to_str:N
7796   \SubMatrix,~type~H~<return>.
7797 }
7798 {
7799   The~available~keys~are~(in~alphabetic~order):~
7800   'delimiters/color',~
7801   'extra-height',~
7802   'hlines',~
7803   'hvlines',~
7804   'left-xshift',~
7805   'name',~
7806   'right-xshift',~
7807   'rules'~(with~the~subkeys~'color'~and~'width'),~
7808   'slim',~
7809   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
7810   and~'right-xshift').\\
7811 }
7812 \@@_msg_new:nnn { Unknown~key~for~notes }
7813 {
7814   The~key~'\l_keys_key_str'~is~unknown.\\
7815   If~you~go~on,~it~will~be~ignored. \\
7816   For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
7817 }
7818 {
7819   The~available~keys~are~(in~alphabetic~order):~
7820   bottomrule,~
7821   code-after,~
7822   code-before,~
7823   enumitem-keys,~
7824   enumitem-keys-para,~
7825   para,~
7826   label-in-list,~
7827   label-in-tabular~and~
7828   style.
7829 }
7830 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
7831 {
7832   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7833   \token_to_str:N \RowStyle. \\
7834   If~you~go~on,~it~will~be~ignored. \\
7835   For~a~list~of~the~available~keys,~type~H~<return>.
7836 }
7837 {
7838   The~available~keys~are~(in~alphabetic~order):~
7839   'bold',~
7840   'cell-space-top-limit',~
7841   'cell-space-bottom-limit',~
7842   'cell-space-limits',~
7843   'color',~
7844   'nb-rows'~and~
7845   'rowcolor'.
7846 }

```



```

7847 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
7848 {
7849   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7850   \token_to_str:N \NiceMatrixOptions. \\
7851   If~you~go~on,~it~will~be~ignored. \\
7852   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7853 }
7854 {
7855   The~available~keys~are~(in~alphabetic~order):~
7856   allow~duplicate~names,~
7857   cell~space~bottom~limit,~
7858   cell~space~limits,~
7859   cell~space~top~limit,~
7860   code~for~first~col,~
7861   code~for~first~row,~
7862   code~for~last~col,~
7863   code~for~last~row,~
7864   corners,~
7865   custom~key,~
7866   create~extra~nodes,~
7867   create~medium~nodes,~
7868   create~large~nodes,~
7869   delimiters~(several~subkeys),~
7870   end~of~row,~
7871   first~col,~
7872   first~row,~
7873   hlines,~
7874   hvlines,~
7875   last~col,~
7876   last~row,~
7877   left~margin,~
7878   letter~for~dotted~lines,~
7879   light~syntax,~
7880   notes~(several~subkeys),~
7881   nullify~dots,~
7882   renew~dots,~
7883   renew~matrix,~
7884   respect~arraystretch,~
7885   right~margin,~
7886   rules~(with~the~subkeys~'color'~and~'width'),~
7887   small,~
7888   sub~matrix~(several~subkeys),
7889   vlines,~
7890   xdots~(several~subkeys).
7891 }
7892 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
7893 {
7894   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7895   \{NiceArray\}. \\
7896   If~you~go~on,~it~will~be~ignored. \\
7897   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7898 }
7899 {
7900   The~available~keys~are~(in~alphabetic~order):~
7901   b,~
7902   baseline,~
7903   c,~
7904   cell~space~bottom~limit,~
7905   cell~space~limits,~
7906   cell~space~top~limit,~
7907   code~after,~
7908   code~for~first~col,~
7909   code~for~first~row,~

```

```

7910 code-for-last-col,~
7911 code-for-last-row,~
7912 colortbl-like,~
7913 columns-width,~
7914 corners,~
7915 create-extra-nodes,~
7916 create-medium-nodes,~
7917 create-large-nodes,~
7918 delimiters/color,~
7919 extra-left-margin,~
7920 extra-right-margin,~
7921 first-col,~
7922 first-row,~
7923 hlines,~
7924 hvlines,~
7925 last-col,~
7926 last-row,~
7927 left-margin,~
7928 light-syntax,~
7929 name,~
7930 notes/bottomrule,~
7931 notes/para,~
7932 nullify-dots,~
7933 renew-dots,~
7934 respect-arraystretch,~
7935 right-margin,~
7936 rules~(with~the~subkeys~'color'~and~'width'),~
7937 small,~
7938 t,~
7939 tabularnote,~
7940 vlines,~
7941 xdots/color,~
7942 xdots/shorten~and~
7943 xdots/line-style.
7944 }

```

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).

```

7945 \@@_msg_new:nnn { Unknown-key-for-NiceMatrix }
7946 {
7947   The~key~'\l_keys_key_str'~is~unknown~for~the~
7948   \@@_full_name_env:. \\\
7949   If~you~go~on,~it~will~be~ignored. \\\
7950   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7951 }
7952 {
7953   The~available~keys~are~(in~alphabetic~order):~
7954   b,~
7955   baseline,~
7956   c,~
7957   cell-space-bottom-limit,~
7958   cell-space-limits,~
7959   cell-space-top-limit,~
7960   code-after,~
7961   code-for-first-col,~
7962   code-for-first-row,~
7963   code-for-last-col,~
7964   code-for-last-row,~
7965   colortbl-like,~
7966   columns-width,~
7967   corners,~
7968   create-extra-nodes,~
7969   create-medium-nodes,~
7970   create-large-nodes,~

```

```

7971 delimiters~(several~subkeys),~
7972 extra-left-margin,~
7973 extra-right-margin,~
7974 first-col,~
7975 first-row,~
7976 hlines,~
7977 hvlines,~
7978 l,~
7979 last-col,~
7980 last-row,~
7981 left-margin,~
7982 light-syntax,~
7983 name,~
7984 nullify-dots,~
7985 r,~
7986 renew-dots,~
7987 respect-arraystretch,~
7988 right-margin,~
7989 rules~(with~the~subkeys~'color'~and~'width'),~
7990 small,~
7991 t,~
7992 vlines,~
7993 xdots/color,~
7994 xdots/shorten~and~
7995 xdots/line-style.
7996 }
7997 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
7998 {
7999   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
8000   \{NiceTabular\}. \\
8001   If~you~go~on,~it~will~be~ignored. \\
8002   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
8003 }
8004 {
8005   The~available~keys~are~(in~alphabetic~order):~
8006   b,~
8007   baseline,~
8008   c,~
8009   cell-space-bottom-limit,~
8010   cell-space-limits,~
8011   cell-space-top-limit,~
8012   code-after,~
8013   code-for-first-col,~
8014   code-for-first-row,~
8015   code-for-last-col,~
8016   code-for-last-row,~
8017   colortbl-like,~
8018   columns-width,~
8019   corners,~
8020   custom-line,~
8021   create-extra-nodes,~
8022   create-medium-nodes,~
8023   create-large-nodes,~
8024   extra-left-margin,~
8025   extra-right-margin,~
8026   first-col,~
8027   first-row,~
8028   hlines,~
8029   hvlines,~
8030   last-col,~
8031   last-row,~
8032   left-margin,~
8033   light-syntax,~

```

```

8034     name,~
8035     notes/bottomrule,~
8036     notes/para,~
8037     nullify-dots,~
8038     renew-dots,~
8039     respect-arraystretch,~
8040     right-margin,~
8041     rules~(with~the~subkeys~'color'~and~'width'),~
8042     t,~
8043     tabularnote,~
8044     vl原因,~
8045     xdots/color,~
8046     xdots/shorten~and~
8047     xdots/line-style.
8048 }

8049 \@@_msg_new:nnn { Duplicate-name }
8050 {
8051     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
8052     the~same~environment~name~twice.~You~can~go~on,~but,~
8053     maybe,~you~will~have~incorrect~results~especially~
8054     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
8055     message~again,~use~the~key~'allow-duplicate-names'~in~
8056     '\token_to_str:N \NiceMatrixOptions'.\\
8057     For~a~list~of~the~names~already~used,~type~H~<return>. \\
8058 }
8059 {
8060     The~names~already~defined~in~this~document~are:~
8061     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
8062 }

8063 \@@_msg_new:nn { Option-auto-for-columns-width }
8064 {
8065     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
8066     If~you~go~on,~the~key~will~be~ignored.
8067 }

```

19 History

The successive versions of the file `nicematrix.sty` provided by TeXLive are available on the SVN server of TeXLive:

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

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types L, C and R.

Changes between version 1.2 and 1.3

New environment `{pNiceArrayC}` and its variants.

Correction of a bug in the definition of `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` (in fact, it was a typo).

Options are now available locally in `{pNiceMatrix}` and its variants.

The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types `w` and `W` can now be used in the environments `{NiceArray}`, `{pNiceArrayC}` and its variants with the same meaning as in the package `array`.

New option `columns-width` to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of `nicematrix` were focused on the continuous dotted lines whereas the version 2.0 of `nicematrix` provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment `{pNiceArrayRC}`. With this new implementation, there is no restriction on the width of the columns.

The package `nicematrix` no longer loads `mathtools` but only `amsmath`.

Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option `code-for-first-row` is now available in the command `\NiceMatrixOptions`.

Following a discussion on TeX StackExchange⁷², Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁷³

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

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 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`).

Possibility to draw vertical dotted lines to separate columns with the specifier “:” in the preamble (similar to the classical specifier “|” and the specifier “:” of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “:” in the preamble.

Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.

Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of “|”) as `\hdotsfor` does.

Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.

Error message when the user gives an incorrect value for `last-row`.

A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol “:” (in the preamble of the array) and `\line` in `code-after`).

The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.

The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by `colortbl`.

Correction of a bug: it was not possible to use the colon “:” in the preamble of an array when `pdf \LaTeX` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

The option `columns-width=auto` doesn’t need any more a second compilation.

The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).

The previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX 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).

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`.

⁷⁴cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -block and, if the creation of the “medium nodes” is required, a node $i-j$ -block-medium is created.

If the user tries to use more columns than allowed by its environment, an error is raised by `nicematrix` (instead of a low-level error).

The package must be loaded with the option `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the “last row”.

The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customisation of the dotted lines.

In the environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, etc.), it’s possible to use the options `l` (=L) or `r` (=R) to specify the type of the columns.

The starred versions of the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots` and `\Iddots` are deprecated since the version 3.1 of `nicematrix`. Now, one should load `nicematrix` with the option `starred-commands` to avoid an error at the compilation.

The code of `nicematrix` no longer uses Tikz but only PGF. By default, Tikz is *not* loaded by `nicematrix`.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on [stackoverflow](#)).

Better error messages when the user uses `&` or `\\` when `light-syntax` is in force.

Changes between versions 3.14 and 3.15

It’s possible to put labels on the dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and the command `\line` in the `code-after` with the tokens `_` and `^`.

The option `baseline` is now available in all the environments of `nicematrix`. Before, it was available only in `{NiceArray}`.

New keyword `\CodeAfter` (in the environments of `nicematrix`).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`

Commands to color cells, rows and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`

New command `\diagbox`

The key `hvline` don’t draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It’s now possible to write `\begin{pNiceMatrix}a&b\\c&d\end{pNiceMatrix}`² with the expected result.

Changes between versions 4.2 and 4.3

The horizontal centering of the content of a `\Block` is correct even when an instruction such as `!\qqquad` is used in the preamble of the array.

It's now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.

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

Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).

Environment `{NiceTabular*}`

Command `\Vdotsfor` similar to `\Hdotsfor`

The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.

Key `respect-blocks` for `\rowcolors` (with a *s*) in the `code-before`.

The variable `\g_nicematrix_code_before_tl` is now public.

The key `baseline` may take in as value an expression of the form *line-i* to align the `\hline` in the row *i*.

The key `hvlines-except-corners` may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys `c`, `r` and `l` for the command `\Block`.

It's possible to use the key `draw-first` with `\Ddots` and `\Iddots` to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key `tabularnote`.

Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put `\omit` before `\CodeAfter`.

Correction of a bug: the tabular notes `\tabularnotes` were not composed when present in a block (except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.

New command `\NotEmpty`.

Changes between versions 5.6 and 5.7

New key `delimiters-color`

Keys `fill`, `draw` and `line-width` for the command `\Block`.

Changes between versions 5.7 and 5.8

Keys `cols` and `restart` of the command `\rowcolors` in the `code-before`.

Modification of the behaviour of `\\` in the columns of type `p`, `m` or `b` (for a behaviour similar to the environments of `array`).

Better error messages for the command `\Block`.

Changes between versions 5.8 and 5.9

Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.

New key `cell-space-limits`.

Changes between versions 5.9 and 5.10

New command `\SubMatrix` available in the `\CodeAfter`.

It's possible to provide options (between brackets) to the keyword `\CodeAfter`.

A (non fatal) error is raised when the key `transparent`, which is deprecated, is used.

Changes between versions 5.10 and 5.11

It's now possible, in the `code-before` and in the `\CodeAfter`, to use the syntax `|(i-|j)` for the Tikz node at the intersection of the (potential) horizontal rule number i and the (potential) vertical rule number j .

Changes between versions 5.11 and 5.12

Keywords `\CodeBefore` and `\Body` (alternative syntax to the key `code-before`).

New key `delimiters/max-width`.

New keys `hlines`, `vlines` and `hvlines` for the command `\SubMatrix` in the `\CodeAfter`.

New key `rounded-corners` for the command `\Block`.

Changes between versions 5.12 and 5.13

New command `\arraycolor` in the `\CodeBefore` (with its key `except-corners`).

New key `borders` for the command `\Block`.

New command `\Hline` (for horizontal rules not drawn in the blocks).

The keys `vlines` and `hlines` takes in as value a (comma-separated) list of numbers (for the rules to draw).

Changes between versions 5.13 and 5.14

Nodes of the form (1.5) , (2.5) , (3.5) , etc.

Keys `t` and `b` for the command `\Block`.

Key `corners`.

Changes between versions 5.14 and 5.15

Key `hvlines` for the command `\Block`.

The commands provided by `nicematrix` to color cells, rows and columns don't color the cells which are in the “corners” (when the key `corner` is used).

It's now possible to specify delimiters for submatrices in the preamble of an environment.

The version 5.15b is compatible with the version 3.0+ of `siunitx` (previous versions were not).

Changes between versions 5.15 and 5.16

It's now possible to use the cells corresponding to the contents of the nodes (of the form `i-j`) in the `\CodeBefore` when the key `create-cell-nodes` of that `\CodeBefore` is used. The medium and the large nodes are also available if the corresponding keys are used.

Changes between versions 5.16 and 5.17

The key `define-L-C-R` (only available at load-time) now raises a (non fatal) error.

Keys `L`, `C` and `R` for the command `\Block`.

Key `hvlines-except-borders`.

It's now possible to use a key `l`, `r` or `c` with the command `\pAutoNiceMatrix` (and the similar ones).

Changes between versions 5.17 and 5.18

New command `\RowStyle`

Changes between versions 5.18 and 5.19

New key `tikz` for the command `\Block`.

Changes between versions 5.19 and 6.0

Columns `X` and environment `{NiceTabularX}`.

Command `\rowlistcolors` available in the `\CodeBefore`.

In columns with fixed width, the blocks are composed as paragraphs (wrapping of the lines).

The key `define-L-C-R` has been deleted.

Changes between versions 6.0 and 6.1

Better computation of the widths of the `X` columns.

Key `\color` for the command `\RowStyle`.

Changes between versions 6.1 and 6.2

Better compatibility with the classes `revtex4-1` and `revtex4-2`.

Key `vlines-in-sub-matrix`.

Changes between versions 6.2 and 6.3

Keys `nb-rows`, `rowcolor` and `bold` for the command `\RowStyle`

Key `name` for the command `\Block`.

Support for the columns `V` of `varwidth`.

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`

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
@@ commands:	
<code>\@@_Block:</code>	1217, 5714
<code>\@@_Block_i</code>	5719, 5720, 5724
<code>\@@_Block_ii:nnnnn</code>	5724, 5725
<code>\@@_Block_iv:nnnnn</code>	5762, 5766
<code>\@@_Block_iv:nnnnnn</code>	5995, 5997
<code>\@@_Block_v:nnnnn</code>	5763, 5889
<code>\@@_Block_v:nnnnnn</code>	6024, 6027
<code>\@@_Cdots</code>	1137, 1208, 3988
<code>\g_@@_Cdots_lines_tl</code>	1238, 3180
<code>\@@_CodeAfter:</code>	1221, 6687
<code>\@@_CodeAfter_i:</code>	870, 2827, 2872, 6688
<code>\@@_CodeAfter_ii:n</code> ..	6687, 6688, 6689, 6699
<code>\@@_CodeAfter_iv:n</code>	6692, 6694
<code>\@@_CodeAfter_keys:</code>	3121, 3140
<code>\@@_CodeBefore:w</code>	1380, 1382
<code>\@@_CodeBefore_keys:</code>	1360, 1377
<code>\@@_Ddots</code>	1139, 1210, 4020
<code>\g_@@_Ddots_lines_tl</code>	1241, 3178
<code>\g_@@_HVDotsfor_lines_tl</code>	1243, 3176, 4104, 4181, 7392
<code>\@@_Hdotsfor:</code>	1142, 1215, 4080
<code>\@@_Hdotsfor:nnnn</code>	4106, 4119
<code>\@@_Hdotsfor_i</code>	4089, 4095, 4102
<code>\@@_Hline:</code>	1213, 5156
<code>\@@_Hline_i:n</code>	5156, 5157, 5163
<code>\@@_Hline_ii:nn</code>	5160, 5163
<code>\@@_Hline_iii:n</code>	5161, 5164
<code>\@@_Hspace:</code>	1214, 4074
<code>\@@_Iddots</code>	1140, 1211, 4044
<code>\g_@@_Iddots_lines_tl</code>	1242, 3179
<code>\@@_Ldots</code>	1136, 1141, 1207, 3972
<code>\g_@@_Ldots_lines_tl</code>	1239, 3181
<code>\l_@@_Matrix_bool</code>	239, 1655, 1679, 2424, 2765, 2771, 2779, 2940
<code>\l_@@_NiceArray_bool</code>	236, 396, 1284, 1523, 1595, 1718, 1725, 1737, 2424, 2753, 2765, 2771, 2779, 2916, 4947, 4951, 5102, 5112, 5142, 5146, 6591
<code>\g_@@_NiceMatrixBlock_int</code>	221, 5472, 5477, 5480, 5491
<code>\l_@@_NiceTabular_bool</code> ...	178, 237, 875, 1063, 1359, 1362, 1490, 1628, 1632, 1726, 1738, 2754, 2971, 2992, 3001, 5796, 5898, 6599
<code>\@@_NotEmpty:</code>	1223, 2963
<code>\@@_OnlyMainNiceMatrix:n</code>	1219, 4734
<code>\@@_OnlyMainNiceMatrix_i:n</code> ..	4737, 4744, 4747
<code>\@@_OverBrace</code>	3115, 7150
<code>\@@_RowStyle:n</code>	1224, 4332
<code>\@@_S:</code>	209, 1768
<code>\@@_SubMatrix</code>	3113, 6840
<code>\@@_SubMatrix_in_code_before</code> ...	1352, 6820
<code>\@@_UnderBrace</code>	3114, 7145
<code>\@@_V:</code>	1683, 1764
<code>\@@_Vdots</code>	1138, 1209, 4004
<code>\g_@@_Vdots_lines_tl</code>	1240, 3177
<code>\@@_Vdotsfor:</code>	1216, 4179
<code>\@@_Vdotsfor:nnnn</code>	4183, 4194
<code>\@@_W:</code>	1682, 1767, 2281
<code>\@@_X</code>	1776, 2987
<code>\l_@@_X_column_bool</code>	241, 2173, 5760

<code>\l_@@_X_columns_aux_bool</code> ..	268, 1548, 2162	<code>\l_@@_borders_tikz_tl</code>	
<code>\l_@@_X_columns_dim</code>	6404, 6445, 6461, 6468, 6481, 6488
.....	269, 1549, 1558, 1564, 2165	<code>\@@_brace:nnnnn</code>	7148, 7153, 7169
<code>\@@_actually_color:</code>	1361, 4419	<code>\l_@@_brace_left_shorten_bool</code>	
<code>\@@_actually_diagbox:nnnnnn</code>	7157, 7188, 7203
.....	5773, 6116, 6611, 6628	<code>\l_@@_brace_right_shorten_bool</code>	
<code>\@@_actually_draw_Cdots:</code>	3546, 3550	7159, 7209, 7224
<code>\@@_actually_draw_Ddots:</code>	3696, 3700	<code>\l_@@_brace_yshift_dim</code> ...	7162, 7245, 7277
<code>\@@_actually_draw_Iddots:</code>	3744, 3748	<code>\@@_cartesian_color:nn</code>	4432, 4444, 4453, 4575
<code>\@@_actually_draw_Ldots:</code> ..	3507, 3511, 4170	<code>\@@_cartesian_path:</code>	4436, 4674
<code>\@@_actually_draw_Vdots:</code> ..	3628, 3632, 4245	<code>\@@_cartesian_path:n</code>	4484, 4616, 4674
<code>\@@_add_to_colors_seq:nn</code>		<code>\@@_cell_begin:w</code>	867, 1813,
.....	4405, 4417, 4418, 4442, 4451, 4460, 4469, 4573	1943, 2014, 2045, 2172, 2290, 2311, 2332, 2949
<code>\@@_adjust_pos_of_blocks_seq:</code> ..	3097, 3142	<code>\l_@@_cell_box</code>	874, 922, 924,
<code>\@@_adjust_pos_of_blocks_seq_i:nnnnn</code>	930, 936, 939, 943, 952, 953, 958, 959, 964,
.....	3145, 3147	965, 976, 977, 978, 979, 981, 984, 988, 990,
<code>\@@_adjust_size_box:</code>	1009, 1024, 1026, 1033, 1034, 1047, 1158,
.....	948, 975, 2023, 2341, 2851, 2896	1268, 1270, 1971, 1975, 1979, 1982, 2013,
<code>\@@_adjust_to_submatrix:nn</code>	2024, 2175, 2331, 2342, 2830, 2854, 2857,
.....	3391, 3494, 3533, 3614, 3689, 3737	2859, 2876, 2899, 2903, 6124, 6245, 6282, 6606
<code>\@@_adjust_to_submatrix:nnnnnn</code> ..	3398, 3400	<code>\@@_cell_end:</code>	969, 1815,
<code>\@@_after_array:</code>	1664, 3005	1962, 2019, 2050, 2181, 2292, 2321, 2337, 2949
<code>\g_@@_after_col_zero_bool</code>		<code>\l_@@_cell_space_bottom_limit_dim</code> ...	
.....	270, 1104, 2828, 4086	479, 559, 979, 4378
<code>\@@_analyze_end:Nn</code>	2572, 2617	<code>\l_@@_cell_space_top_limit_dim</code>	
<code>\l_@@_argspec_tl</code> ..	3970, 3971, 3972, 3988,	478, 557, 977, 4367
.....	4004, 4020, 4044, 4100, 4101, 4102, 4177,	<code>\@@_cellcolor</code> ..	1343, 4486, 4498, 4499, 4705
.....	4178, 4179, 4255, 4256, 4257, 6838, 6839, 6840	<code>\@@_cellcolor_tabular</code>	1146, 4699
<code>\@@_array:</code>	1061, 2573, 2600	<code>\g_@@_cells_seq</code>	2611, 2612, 2613, 2615
<code>\@@_arraycolor</code>	1349, 4504	<code>\@@_center_cell_box:</code>	1921, 1966
<code>\c_@@_arydshln_loaded_bool</code> ...	21, 33, 1793	<code>\@@_chessboardcolors</code>	1351, 4491
<code>\l_@@_auto_columns_width_bool</code>		<code>\@@_cline</code>	151, 1206
.....	504, 659, 2684, 2688, 5467	<code>\@@_cline_i:nn</code>	152, 153, 173, 176
<code>\g_@@_aux_found_bool</code>		<code>\@@_cline_i:w</code>	153, 154
.....	1255, 1260, 1384, 1511, 1514	<code>\@@_cline_ii:w</code>	158, 160
<code>\g_@@_aux_tl</code>	242,	<code>\@@_cline_iii:w</code>	157, 161, 162
.....	1517, 1546, 1671, 3014, 3028, 3036, 3129, 5354	<code>\l_@@_code_before_bool</code>	
<code>\l_@@_bar_at_end_of_pream_bool</code>	274, 649, 676, 1077, 1275, 1306, 1520,
.....	1719, 1862, 2757	2631, 2648, 2666, 2697, 2723, 2760, 2799, 3134
<code>\l_@@_baseline_tl</code> ..	493, 494, 652, 653, 654,	<code>\g_@@_code_before_tl</code> ..	1509, 1518, 1521, 3131
.....	655, 1070, 1597, 2377, 2389, 2394, 2396,	<code>\l_@@_code_before_tl</code>	
.....	2401, 2406, 2496, 2497, 2501, 2506, 2508, 2513	273, 648, 1305, 1360, 1521
<code>\@@_begin_of_NiceMatrix:nn</code>	2938, 2959	<code>\l_@@_code_for_first_col_tl</code>	576, 2841
<code>\@@_begin_of_row:</code>	873, 898, 1167, 2829	<code>\l_@@_code_for_first_row_tl</code> ..	580, 886, 6214
<code>\l_@@_block_auto_columns_width_bool</code> ..		<code>\l_@@_code_for_last_col_tl</code>	578, 2885
.....	1503, 2689, 5460, 5465, 5475, 5485	<code>\l_@@_code_for_last_row_tl</code> ..	582, 893, 6217
<code>\g_@@_block_box_int</code>	315, 1484,	<code>\l_@@_code_tl</code>	262, 6815, 7051
.....	5768, 5782, 5784, 5842, 5854, 5866, 5875, 5885	<code>\l_@@_col_max_int</code>	
<code>\l_@@_block_name_str</code>	297, 3257, 3268, 3336, 3396, 3413
.....	5982, 6063, 6140, 6143, 6148	<code>\l_@@_col_min_int</code>	
<code>\@@_block_tikz:nnnnn</code>	6104, 6499	296, 3262, 3325, 3330, 3394, 3411
<code>\g_@@_blocks_dp_dim</code>		<code>\g_@@_col_total_int</code>	
.....	232, 956, 959, 960, 5869, 5872	226, 992, 1234, 1314, 1325,
<code>\g_@@_blocks_ht_dim</code>	1397, 1581, 2262, 2263, 2716, 2717, 2747,
.....	231, 962, 965, 966, 5860, 5863	2802, 2807, 2814, 2816, 2875, 3009, 3011,
<code>\g_@@_blocks_seq</code>	3023, 3581, 3599, 3659, 4240, 4724, 5518,
.....	285, 1505, 2434, 5879, 5891, 5995	5528, 5562, 5649, 6007, 6252, 6876, 6925, 7175
<code>\g_@@_blocks_wd_dim</code>		<code>\l_@@_col_width_dim</code>	
.....	230, 950, 953, 954, 5848, 5851	223, 224, 1942, 2012, 5801, 5806
<code>\c_@@_booktabs_loaded_bool</code> ..	22, 36, 1156, 2467	<code>\@@_color_index:n</code>	4563, 4584, 4587
<code>\l_@@_borders_clist</code>	303, 5954,	<code>\l_@@_color_int</code>	
.....	6077, 6407, 6432, 6434, 6436, 6438, 6475, 6494	4527, 4528, 4545, 4546, 4566, 4577

\l_@@_color_tl	\g_@@_delta_x_two_dim
305, 4560, 4561, 4571, 4574, 5709, 5786, 5788	3080, 3767, 3777
\g_@@_colors_seq 1356, 4408, 4412, 4413, 4423	\g_@@_delta_y_one_dim
\l_@@_colors_seq 4522, 4523, 4567, 4586, 4588	3079, 3721, 3729
\@@_colortbl_like:	\g_@@_delta_y_two_dim
1144, 1225	3081, 3769, 3777
\l_@@_colortbl_like_bool 476, 675, 1225, 1707	\@@_diagbox:nn
\c_@@_colortbl_loaded_bool . 105, 109, 1175	1222, 6607
\l_@@_cols_tl	\@@_dotfill:
4435, 4483, 4514, 4524, 4525, 4575, 4622, 4625	6596
\g_@@_cols_vlism_seq	\@@_dotfill_i:
250, 1273, 1702, 1785, 6953	6601, 6603
\@@_columncolor	\@@_dotfill_ii:
1350, 4447	6600, 6603, 6604
\@@_columncolor_preamble	\@@_dotfill_iii:
1148, 4722	6604, 6605
\c_@@_columncolor_regex	\l_@@_dotted_bool
64, 1710	323, 3789, 4766, 4852, 4854, 5033, 5035
\l_@@_columns_width_dim	\@@_double_int_eval:n
222, 660, 783, 2685, 2691, 5473, 5479	4251, 4265, 4266
\g_@@_com_or_env_str	\g_@@_dp_ante_last_row_dim
254, 257	901, 1191
\l_@@_command_str 5184, 5205, 5225, 5248, 5262	\g_@@_dp_last_row_dim
\@@_computations_for_large_nodes: ...	901, 902, 1194, 1195, 1269, 1270, 1614
5589, 5602, 5607	\g_@@_dp_row_zero_dim
\@@_computations_for_medium_nodes: ..	921, 922, 1185, 1186, 1607, 2490, 2529
5509, 5578, 5588, 5599	\@@_draw_Cdots:nnn
\@@_compute_a_corner:nnnnnn	3531
5342, 5344, 5346, 5348, 5361	\@@_draw_Ddots:nnn
\@@_compute_corners:	3687
3096, 5334	\@@_draw_Iddots:nnn
\@@_compute_i_j:nn	3735
6849, 6873, 7172	\@@_draw_Ldots:nnn
\@@_construct_preamble:	3492
1281, 1676	\@@_draw_Vdots:nnn
\l_@@_corners_cells_seq	3612
289, 4619, 4659, 4830, 4836, 4843,	\@@_draw_blocks:
5011, 5017, 5024, 5336, 5352, 5356, 5357, 5417	2434, 5992
\l_@@_corners_clist	\@@_draw_dotted_lines:
498, 637, 642, 4804, 4985, 5337	3095, 3165
\@@_create_blocks_nodes:	\@@_draw_dotted_lines_i:
1334, 1429	3168, 3172
\@@_create_col_nodes:	\l_@@_draw_first_bool . 312, 4035, 4059, 4070
2576, 2604, 2623	\@@_draw_hlines:
\@@_create_diag_nodes: ...	3098, 5138
1331, 3044, 3198	\@@_draw_line:
\@@_create_extra_nodes: ..	3529,
1427, 2433, 5499	3574, 3685, 3733, 3781, 3783, 4304, 4923, 5118
\@@_create_large_nodes:	\@@_draw_line_ii:nn
5507, 5583	4284, 4288
\@@_create_medium_and_large_nodes: ..	\@@_draw_line_iii:nn
5504, 5594	4291, 4295
\@@_create_medium_nodes:	\@@_draw_standard_dotted_line: . 3790, 3821
5505, 5573	\@@_draw_standard_dotted_line_i: 3884, 3888
\@@_create_nodes: 5580, 5591, 5601, 5604, 5645	\l_@@_draw_tl
\@@_create_one_block_node:nnnnn 1435, 1438	301, 5948,
\@@_create_row_node:	5952, 6065, 6297, 6303, 6305, 6307, 6344, 6345
1073, 1107, 1157	\@@_draw_unstandard_dotted_line: 3791, 3793
\@@_custom_line:n	\@@_draw_unstandard_dotted_line:n ...
550, 5182	3796, 3799
\@@_custom_line_i:n	\@@_draw_unstandard_dotted_line:nnn ..
5219, 5272	3801, 3806, 3820
\@@_cut_on_hyphen:w	\@@_draw_vlines:
335,	3099, 4943
348, 4476, 4481, 4541, 4629, 4630, 4652,	\g_@@_empty_cell_bool
4653, 4683, 4684, 6315, 6324, 6355, 6358,	282, 983, 993,
6378, 6381, 6408, 6411, 6824, 6829, 6855, 6862	2864, 2911, 3986, 4002, 4018, 4042, 4065, 4076
\g_@@_ddots_int	\l_@@_end_int
3076, 3716, 3717	4758,
\@@_def_env:nnn	4782, 4783, 4794, 4821, 4964, 4965, 4975, 5002
2922, 2933, 2934, 2935, 2936, 2937	\l_@@_end_of_row_tl
\@@_define_com:nnn	514, 515, 570, 2596, 2597, 7502
6575, 6583, 6584, 6585, 6586, 6587	\c_@@_endpgfortikzpicture_tl
\@@_define_h_custom_line:nn	48, 52, 3169, 4292, 6423
5235, 5260	\c_@@_enumitem_loaded_bool
\@@_delimiter:nnn	23, 39, 369, 701, 710
2078, 2099, 2107, 2120, 2126, 2132, 6702	\@@_env:
\l_@@_delimiters_color_tl	216, 220, 907, 913,
517,	1010, 1016, 1030, 1038, 1083, 1089, 1095,
721, 1373, 1620, 1621, 1638, 1639, 6681,	1170, 1315, 1316, 1321, 1322, 1327, 1328,
6740, 6741, 6768, 6790, 7103, 7104, 7127, 7128	1340, 1394, 1395, 1400, 1403, 1406, 1409,
\l_@@_delimiters_max_width_bool	1418, 1419, 1424, 1425, 1451, 2632, 2635,
518, 719, 1643	2637, 2653, 2659, 2662, 2671, 2677, 2680,
\g_@@_delta_x_one_dim	2702, 2708, 2711, 2729, 2735, 2741, 2768,
3078, 3719, 3729	2777, 2791, 2802, 2807, 2816, 3049, 3050,
	3055, 3056, 3064, 3070, 3108, 3215, 3217,
	3224, 3226, 3227, 3232, 3235, 3297, 3365,
	3430, 3441, 3454, 3457, 3476, 3479, 3584,
	3587, 3602, 3605, 4133, 4151, 4208, 4226,
	4277, 4279, 4298, 4301, 5372, 5391, 5409,

5531, 5533, 5541, 5652, 5661, 5679, 6138,
6143, 6144, 6149, 6158, 6162, 6176, 6181,
6193, 6199, 6200, 6203, 6222, 6259, 6717,
6720, 6892, 6894, 6899, 6901, 6928, 6930,
6935, 6937, 7035, 7047, 7194, 7196, 7215, 7217
\g_@@_env_int
. 215, 216, 218, 1502, 1512, 1515, 1670, 5688
\@@_error:n 12,
372, 397, 527, 539, 592, 632, 715, 773, 782,
791, 799, 817, 824, 832, 833, 834, 840, 845,
846, 847, 861, 863, 864, 865, 1375, 1542,
1576, 1586, 1658, 2003, 2411, 2472, 2518,
4330, 4517, 5194, 5196, 5201, 5206, 5214,
5223, 5944, 5990, 6035, 6402, 6446, 6452,
6685, 6800, 6811, 6818, 7167, 7327, 7354, 7364
\@@_error:nn ... 13, 667, 2081, 2127, 2133,
2157, 3975, 3978, 3991, 3994, 4007, 4010,
4024, 4025, 4030, 4031, 4048, 4049, 4054,
4055, 5350, 6805, 6877, 6907, 6910, 7178, 7179
\@@_error:nnn 14, 4282, 6982, 7017
\@@_error_too_much_cols: 1747, 7377
\@@_everycr: 1100, 1180, 1183
\@@_everycr_i: 1100, 1101
\l_@@_except_borders_bool
..... 509, 604, 4947, 4951, 5142, 5146
\@@_exec_code_before: 1275, 1354
\@@_expand_clist:N 340, 1154, 1155
\@@_expand_clist:NN 4622, 4623, 4675
\l_@@_exterior_arraycolsep_bool
..... 495, 779, 1728, 1740, 2756
\l_@@_extra_left_margin_dim
..... 512, 620, 1297, 2862
\l_@@_extra_right_margin_dim
..... 513, 621, 1537, 2907, 3662
\@@_extract_brackets 6089, 6287
\@@_extract_coords_values: 5670, 5677
\@@_fatal:n 15,
246, 1493, 1798, 2055, 2059, 2088, 2581,
2585, 2587, 2620, 4091, 7323, 7382, 7385, 7388
\@@_fatal:nn 16, 1805, 2284
\l_@@_fill_tl 300, 5946, 6087, 6089
\l_@@_final_i_int
..... 3085, 3244, 3249, 3252, 3277,
3285, 3289, 3298, 3306, 3386, 3442, 3523,
3596, 3602, 3605, 4124, 4152, 4220, 4230, 4232
\l_@@_final_j_int
..... 3086, 3245, 3250, 3257, 3262, 3268, 3278,
3286, 3290, 3299, 3307, 3385, 3387, 3443,
3476, 3479, 3487, 4145, 4155, 4157, 4199, 4228
\l_@@_final_open_bool
..... 3088, 3251, 3255, 3258, 3265, 3271, 3275,
3291, 3520, 3555, 3560, 3571, 3635, 3645,
3650, 3671, 3708, 3756, 3892, 3907, 3938,
3939, 4122, 4146, 4158, 4197, 4221, 4233, 4274
\@@_find_extremities_of_line:nnnn ...
..... 3239, 3497, 3536, 3617, 3692, 3740
\l_@@_first_col_int
..... 139, 152, 326, 327, 572, 838,
873, 1325, 1397, 1590, 1720, 2626, 2646,
3021, 3581, 3599, 4084, 4643, 4736, 5518,
5528, 5562, 5610, 5649, 6556, 6562, 6568, 6925
\l_@@_first_i_tl 6851,
6857, 6858, 6888, 6919, 6928, 6930, 6985,
6992, 6994, 7076, 7087, 7091, 7191, 7212, 7240
\l_@@_first_j_tl
..... 6852, 6860, 6861, 6892, 6894, 6955, 6968,
6975, 6977, 7077, 7088, 7092, 7194, 7196, 7206
\l_@@_first_row_int .. 324, 325, 573, 842,
1232, 1319, 1392, 1605, 2408, 2487, 2515,
2526, 3018, 3451, 3473, 5511, 5525, 5552,
5609, 5647, 6155, 6173, 6554, 6714, 6889, 7513
\c_@@_footnote_bool
..... 1479, 1674, 7303, 7325, 7348, 7351, 7361, 7367
\c_@@_footnotehyper_bool . 7302, 7326, 7358
\c_@@_forbidden_letters_str 5213, 5241
\@@_full_name_env:
..... 255, 7403, 7410, 7418, 7433, 7437, 7527,
7545, 7576, 7589, 7594, 7599, 7601, 7634,
7653, 7658, 7663, 7669, 7678, 7723, 7732, 7948
\@@_hdottedline: 1212, 5446
\@@_hdottedline:n 5454, 5456
\@@_hdottedline_i: 5449, 5451
\@@_hline:n 4961, 5153, 5173, 5268, 5457, 6388
\@@_hline_i: 4967, 4970
\@@_hline_ii: 4995, 5003, 5031
\@@_hline_iii: 5039, 5043
\@@_hline_iv: 5036, 5089
\@@_hline_v: 5040, 5121
\@@_hlines_block:nnn 6053, 6374
\l_@@_hlines_block_bool 314, 5959, 6049, 6060
\l_@@_hlines_clist .. 319, 584, 598, 603,
639, 1108, 1110, 1114, 1154, 3098, 5151, 5152
\l_@@_hpos_block_str 307, 308, 5693,
5695, 5697, 5699, 5701, 5703, 5740, 5741,
5743, 5795, 5807, 5820, 5831, 5882, 5906,
5910, 5923, 5928, 5963, 5965, 5967, 5969,
5972, 5975, 6226, 6238, 6249, 6253, 6263, 6275
\l_@@_hpos_cell_str 228, 229,
1813, 1909, 1911, 2015, 2290, 2333, 5739, 5741
\l_@@_hpos_col_str
..... 1865, 1867, 1869, 1871, 1896, 1908,
1912, 1914, 1922, 1923, 1926, 1931, 1998, 2149
\l_@@_hpos_of_block_cap_bool
..... 309, 5970, 5973, 5976, 6152, 6223, 6260
\g_@@_ht_last_row_dim
..... 903, 1192, 1193, 1267, 1268, 1613
\g_@@_ht_row_one_dim .. 929, 930, 1189, 1190
\g_@@_ht_row_zero_dim
..... 923, 924, 1187, 1188, 1608, 2489, 2528
\@@_i: 5511, 5513,
5514, 5515, 5516, 5525, 5531, 5533, 5534,
5535, 5536, 5541, 5542, 5543, 5544, 5552,
5555, 5557, 5558, 5559, 5611, 5613, 5616,
5617, 5621, 5622, 5647, 5652, 5654, 5656,
5660, 5661, 5672, 5679, 5681, 5683, 5687, 5688
\g_@@_iddots_int 3077, 3764, 3765
\l_@@_in_env_bool 235, 396, 1493, 1494
\l_@@_initial_i_int 3083,
3242, 3317, 3320, 3345, 3353, 3357, 3366,
3374, 3384, 3431, 3516, 3562, 3564, 3578,
3584, 3587, 4123, 4124, 4134, 4202, 4212, 4214
\l_@@_initial_j_int
..... 3084, 3243, 3318, 3325,
3330, 3336, 3346, 3354, 3358, 3367, 3375,
3385, 3387, 3432, 3454, 3457, 3465, 3652,
3654, 3659, 4127, 4137, 4139, 4198, 4199, 4210

<code>\l_@@_initial_open_bool</code>	<code>\g_@@_left_delim_tl</code>
..... 3087, 3319, 3323, 3326,	1290, 1481, 1622, 1646, 1716, 2062, 2064, 5104
3333, 3339, 3343, 3359, 3513, 3552, 3559,	<code>\l_@@_left_margin_dim</code>
3569, 3635, 3642, 3648, 3702, 3750, 3890, 510, 614, 1296, 2861, 5101, 5640
3937, 4121, 4128, 4140, 4196, 4203, 4215, 4273	<code>\l_@@_letter_for_dotted_lines_str</code> ...
<code>\@@_insert_tabularnotes:</code> 2439, 2442 790, 801, 802, 1782, 7672
<code>\@@_instruction_of_type:nnn</code>	<code>\l_@@_letter_str</code> 5185,
..... 1050, 3980, 3996, 4012, 4035, 4059	5204, 5208, 5210, 5213, 5218, 5245, 7560, 7571
<code>\c_@@_integers_alist_tl</code> 7061, 7072	<code>\l_@@_letter_vlism_tl</code> 249, 591, 1783
<code>\l_@@_inter_dots_dim</code>	<code>\l_@@_light_syntax_bool</code> 492, 568, 1299, 1532
..... 480, 482, 3092, 3895, 3902,	<code>\@@_light_syntax_i</code> 2589, 2592
3913, 3921, 3928, 3933, 3945, 3953, 5105, 5115	<code>\@@_line</code> 3116, 4257
<code>\g_@@_internal_code_after_tl</code>	<code>\@@_line_i:nn</code> 4264, 4271
..... 263, 1849, 2077, 2098,	<code>\l_@@_line_width_dim</code>
2106, 2119, 2125, 2131, 2192, 3117, 3118,	306, 5961, 6298, 6336, 6347, 6353, 6376,
5171, 5267, 5287, 5453, 5771, 6114, 6609, 6832	6399, 6431, 6457, 6459, 6476, 6477, 6479, 6497
<code>\@@_intersect_our_row:nnnnn</code> 4605	<code>\@@_line_with_light_syntax:n</code> ... 2603, 2607
<code>\@@_intersect_our_row_p:nnnnn</code> 4555	<code>\@@_line_with_light_syntax_i:n</code>
<code>\@@_j:</code> 5518, 5520, 2602, 2608, 2609
5521, 5522, 5523, 5528, 5531, 5533, 5536,	<code>\l_@@_local_end_int</code>
5538, 5539, 5541, 5544, 5546, 5547, 5562, 4792, 4813, 4821, 4871, 4920,
5565, 5567, 5568, 5569, 5624, 5626, 5629,	4935, 4973, 4994, 5002, 5052, 5107, 5109, 5130
5631, 5635, 5636, 5649, 5652, 5653, 5655,	<code>\l_@@_local_start_int</code>
5660, 5661, 5673, 5679, 5680, 5682, 5687, 5688 4791, 4807, 4808, 4811,
<code>\l_@@_key_nb_rows_int</code>	4815, 4819, 4867, 4918, 4931, 4972, 4988,
..... 227, 4324, 4344, 4352, 4359	4989, 4992, 4996, 5000, 5048, 5097, 5099, 5126
<code>\l_@@_l_dim</code>	<code>\@@_math_toggle_token:</code> 177, 971,
.... 3868, 3869, 3882, 3883, 3895, 3901,	2831, 2848, 2877, 2893, 6654, 6665, 7258, 7294
3912, 3920, 3928, 3933, 3945, 3946, 3953, 3954	<code>\g_@@_max_cell_width_dim</code>
<code>\l_@@_large_nodes_bool</code> 508, 611, 5503, 5507 980, 981, 1504, 2690, 5466, 5492
<code>\g_@@_last_col_found_bool</code> 334,	<code>\c_@@_max_l_dim</code> 3882, 3887
1237, 1582, 1650, 2715, 2794, 2873, 3008, 6250	<code>\l_@@_medium_nodes_bool</code> 507, 610, 5501, 6196
<code>\l_@@_last_col_int</code>	<code>\@@_message_hdotsfor:</code> 7390, 7403, 7410, 7418
.. 332, 333, 774, 810, 812, 825, 841, 862,	<code>\@@_msg_new:nn</code> 17, 7330, 7339, 7395,
1258, 1261, 1585, 1732, 2945, 2947, 3009,	7400, 7407, 7415, 7423, 7430, 7435, 7441,
3011, 3622, 3657, 3977, 3993, 4031, 4055,	7446, 7451, 7456, 7461, 7466, 7471, 7476,
6000, 6005, 6006, 6007, 6010, 6046, 6056,	7482, 7488, 7493, 7499, 7505, 7510, 7517,
6072, 6084, 6096, 6108, 6120, 6135, 6176,	7524, 7530, 7539, 7544, 7546, 7551, 7558,
6181, 6189, 6205, 6558, 6564, 6570, 7381, 7404	7563, 7568, 7574, 7580, 7597, 7604, 7609,
<code>\l_@@_last_col_without_value_bool</code> ...	7616, 7626, 7632, 7639, 7646, 7651, 7661,
..... 331, 809, 3010, 7384	7667, 7675, 7682, 7689, 7695, 7701, 7708,
<code>\l_@@_last_empty_column_int</code>	7714, 7720, 7728, 7730, 7736, 7741, 7758, 8063
..... 5382, 5383, 5396, 5402, 5415	<code>\@@_msg_new:nnn</code>
<code>\l_@@_last_empty_row_int</code> 18, 7304, 7586, 7747, 7764, 7776,
..... 5364, 5365, 5378, 5399	7791, 7812, 7830, 7847, 7892, 7945, 7997, 8049
<code>\l_@@_last_i_tl</code>	<code>\@@_msg_redirect_name:nn</code>
.... 6853, 6864, 6865, 6875, 6888, 6922, 19, 635, 785, 1662, 6013, 6016
6935, 6937, 6985, 6992, 7174, 7191, 7212, 7272	<code>\@@_multicolumn:nnn</code> 1227, 2222
<code>\l_@@_last_j_tl</code>	<code>\g_@@_multicolumn_cells_seq</code>
..... 6854, 6867, 6868, 6876, 6899, 292, 1230, 1277, 2237,
6901, 6958, 6968, 6975, 7175, 7215, 7217, 7227	3034, 3038, 3039, 5536, 5544, 5666, 6160, 6178
<code>\l_@@_last_row_int</code>	<code>\g_@@_multicolumn_sizes_seq</code>
328, 329, 574, 891, 937, 1120, 1252, 1256, 293, 1231, 1278, 2239, 3040, 3041, 5667
1263, 1570, 1574, 1577, 1589, 1611, 2598,	<code>\l_@@_multiplicity_int</code>
2599, 2837, 2838, 2882, 2883, 3013, 3502,	... 4764, 4875, 4887, 4897, 5056, 5066, 5076
3541, 4009, 4025, 4049, 4742, 4750, 5999,	<code>\g_@@_name_env_str</code> 253, 258,
6002, 6003, 6022, 6046, 6056, 6072, 6084,	259, 1488, 1489, 2619, 2917, 2918, 2926,
6096, 6107, 6119, 6133, 6204, 6216, 6566, 7722	2927, 2956, 2969, 2988, 2998, 3136, 6579, 7379
<code>\l_@@_last_row_without_value_bool</code> ...	<code>\l_@@_name_str</code>
..... 330, 1254, 1572, 3012	... 506, 669, 909, 912, 1012, 1015, 1091,
<code>\l_@@_left_delim_dim</code>	1094, 2636, 2637, 2661, 2662, 2679, 2680,
..... 1282, 1286, 1291, 2564, 2860	2710, 2711, 2737, 2740, 2787, 2790, 2809,

2813, 3058, 3063, 3069, 3216, 3217, 3228,
 3231, 3234, 5657, 5660, 5684, 5687, 6145, 6148
 \g_@@_names_seq 234, 666, 668, 8061
 \l_@@_nb_cols_int
 6532, 6543, 6551, 6557, 6563, 6569
 \l_@@_nb_rows_int 6531, 6542, 6560
 \@@_newcolumntype 1127, 1681, 1682, 1683, 2975
 \@@_node_for_cell: 989, 996, 1365, 2858, 2908
 \@@_node_for_multicolumn:nn 5668, 5675
 \@@_node_left:nn 7034, 7035, 7095
 \@@_node_position:
 1321, 1323, 1327, 1329, 1394, 1396, 1404, 1410
 \@@_node_position_i: 1407, 1411
 \@@_node_right:nnnn 7044, 7046, 7119
 \g_@@_not_empty_cell_bool 272, 987, 994, 2964
 \@@_not_in_exterior:nnnnn 4597
 \@@_not_in_exterior_p:nnnnn 4533
 \l_@@_notes_above_space_dim 499, 501
 \l_@@_notes_bottomrule_bool
 687, 828, 856, 2465
 \l_@@_notes_code_after_tl 685, 2474
 \l_@@_notes_code_before_tl 683, 2446
 \@@_notes_label_in_list:n 365, 384, 392, 695
 \@@_notes_label_in_tabular:n . 364, 405, 692
 \l_@@_notes_para_bool .. 681, 826, 854, 2450
 \@@_notes_style:n
 363, 366, 384, 392, 408, 413, 689
 \l_@@_nullify_dots_bool
 502, 609, 3984, 4000, 4016, 4040, 4063
 \l_@@_number_of_notes_int 362, 399, 409, 419
 \@@_old_CT@arc@ 1495, 3138
 \@@_old_cdots 1200, 4001
 \@@_old_ddots 1202, 4041
 \@@_old_dotfill 6595, 6598, 6606
 \@@_old_dotfill: 1220
 \l_@@_old_iRow_int 264, 1248, 3185
 \@@_old_ialign: 1072, 1196, 3112, 5994
 \@@_old_iddots 1203, 4064
 \l_@@_old_jCol_int 265, 1250, 3186
 \@@_old_ldots 1199, 3985
 \@@_old_multicolumn 1229, 4079
 \@@_old_pgfpntanchor 191, 7053, 7057
 \@@_old_pgful@check@rerun 98, 102
 \@@_old_vdots 1201, 4017
 \@@_open_x_final_dim:
 3470, 3522, 3556, 3710, 3759
 \@@_open_x_initial_dim:
 3448, 3515, 3553, 3705, 3753
 \@@_open_y_final_dim: 3594, 3646, 3758
 \@@_open_y_initial_dim:
 3576, 3643, 3704, 3752
 \l_@@_other_keys_tl .. 4784, 4853, 4966, 5034
 \@@_overbrace_i:n 7233, 7238
 \l_@@_parallelize_diags_bool
 496, 497, 606, 3074, 3714, 3762
 \@@_patch_for_revtext: 1458, 1477
 \@@_patch_m_preamble:n
 2231, 2266, 2299, 2304, 2365
 \@@_patch_m_preamble_i:n
 2270, 2271, 2272, 2286
 \@@_patch_m_preamble_ii:nn
 2273, 2274, 2275, 2296
 \@@_patch_m_preamble_iii:n 2276, 2301
 \@@_patch_m_preamble_iv:nnn
 2277, 2278, 2279, 2306
 \@@_patch_m_preamble_ix:n 2350, 2368
 \@@_patch_m_preamble_v:nnnn 2280, 2281, 2326
 \@@_patch_m_preamble_x:n
 2294, 2324, 2345, 2347, 2371
 \@@_patch_node_for_cell: 1022, 1365
 \@@_patch_node_for_cell:n 1020, 1046, 1049
 \@@_patch_preamble:n
 1704, 1750, 1789, 1796, 1804, 1823, 1859,
 2066, 2082, 2084, 2100, 2108, 2134, 2194, 2214
 \@@_patch_preamble_i:n 1754, 1755, 1756, 1809
 \@@_patch_preamble_ii:nn
 1757, 1758, 1759, 1820
 \@@_patch_preamble_iii:n . 1760, 1825, 1833
 \@@_patch_preamble_iii_i:n 1828, 1830
 \@@_patch_preamble_iv:n
 1761, 1762, 1763, 1881
 \@@_patch_preamble_iv_i:n 1884, 1886
 \@@_patch_preamble_iv_ii:w 1889, 1890, 1892
 \@@_patch_preamble_iv_iii:nn ... 1893, 1894
 \@@_patch_preamble_iv_iv:nn
 1898, 1900, 2001, 2004, 2164
 \@@_patch_preamble_iv_v:nnnnnnnn 1904, 1937
 \@@_patch_preamble_ix:nn
 1772, 1773, 1774, 2086
 \@@_patch_preamble_ix_i:nnn 2090, 2112
 \@@_patch_preamble_v:n ... 1764, 1765, 1987
 \@@_patch_preamble_v_i:w . 1990, 1991, 1993
 \@@_patch_preamble_v_ii:nn 1994, 1995
 \@@_patch_preamble_vi:nnnn 1766, 1767, 2007
 \@@_patch_preamble_vii:n 1768, 2030
 \@@_patch_preamble_vii_i:w 2033, 2034, 2036
 \@@_patch_preamble_vii_ii:n 2037, 2038
 \@@_patch_preamble_viii:nn
 1769, 1770, 1771, 2057
 \@@_patch_preamble_viii_i:nn
 2070, 2073, 2075
 \@@_patch_preamble_x:n ... 1775, 1776, 2137
 \@@_patch_preamble_x_i:w . 2140, 2141, 2143
 \@@_patch_preamble_x_ii:n 2144, 2147
 \@@_patch_preamble_xi:n
 ... 1818, 1935, 2028, 2053, 2185, 2196, 2220
 \@@_patch_preamble_xii:n 1782, 2188
 \@@_patch_preamble_xiii:n 2199, 2217
 \@@_pgf_rect_node:nnn 452, 1408, 6198
 \@@_pgf_rect_node:nnnnn
 427, 1450, 5651, 5678, 6137, 6192, 7021
 \c_@@_pgfortikzpicture_tl
 47, 51, 3167, 4290, 6421
 \@@_pgfpntanchor:n 7049, 7054
 \@@_pgfpntanchor_i:nn 7057, 7059
 \@@_pgfpntanchor_ii:w 7060, 7068
 \@@_pgfpntanchor_iii:w 7081, 7083
 \@@_picture_position:
 1316, 1323, 1329, 1396, 1410, 1411
 \g_@@_pos_of_blocks_seq
 286, 1276, 1434, 1506, 2240, 3026, 3030,
 3031, 3144, 4531, 4798, 4979, 5429, 6062, 6619
 \g_@@_pos_of_stroken_blocks_seq
 288, 1507, 4802, 4983, 6074
 \g_@@_pos_of_xdots_seq
 287, 1508, 3382, 4800, 4981

```

\l_@@_position_int ..... 4754, 4785,
4793, 4869, 4915, 4933, 4974, 5050, 5094, 5128
\g_@@_post_action_cell_tl .....
..... 869, 973, 1968, 2174, 4365, 4376
\@@_pre_array: ..... 1246, 1307, 1529
\@@_pre_array_i:w ..... 1303, 1529
\@@_pre_array_ii: ..... 1150, 1279
\@@_pre_code_before: ..... 1309, 1386
\c_@@_preamble_first_col_tl .... 1721, 2823
\c_@@_preamble_last_col_tl .... 1733, 2868
\g_@@_preamble_tl .....
.... 1483, 1684, 1688, 1692, 1698, 1712,
1721, 1730, 1733, 1742, 1746, 1787, 1795,
1811, 1822, 1835, 1939, 2009, 2042, 2069,
2097, 2105, 2118, 2124, 2169, 2190, 2203,
2210, 2219, 2230, 2232, 2288, 2298, 2303,
2308, 2328, 2354, 2361, 2370, 2573, 2600, 5282
\@@_provide_pgfsyspdfmark: ... 65, 74, 1478
\@@_put_box_in_flow: ..... 1648, 2373, 2566
\@@_put_box_in_flow_bis:nn .... 1645, 2533
\@@_put_box_in_flow_i: ..... 2379, 2381
\@@_qpoint:n ..... 219, 1442, 1444,
1446, 1448, 2384, 2386, 2398, 2414, 2481,
2483, 2499, 2510, 2521, 3204, 3206, 3208,
3210, 3220, 3222, 3465, 3487, 3516, 3523,
3562, 3564, 3578, 3596, 3652, 3654, 4298,
4301, 4642, 4646, 4662, 4664, 4867, 4869,
4871, 4915, 4918, 4920, 4931, 4933, 4935,
5048, 5050, 5052, 5094, 5097, 5107, 5126,
5128, 5130, 5557, 5567, 6129, 6131, 6133,
6135, 6169, 6189, 6219, 6320, 6322, 6329,
6331, 6456, 6458, 6460, 6474, 6478, 6480,
6633, 6635, 6638, 6640, 6707, 6709, 6919,
6922, 6960, 6977, 6994, 7206, 7227, 7240, 7272
\l_@@_radius_dim .....
486, 488, 2191, 3091, 3527, 3528, 3962, 5448
\l_@@_real_left_delim_dim 2535, 2550, 2565
\l_@@_real_right_delim_dim 2536, 2562, 2568
\@@_recreate_cell_nodes: ..... 1332, 1390
\g_@@_recreate_cell_nodes_bool .....
..... 505, 1165, 1332, 1357, 1364, 1369
\@@_rectanglecolor .....
..... 1344, 4339, 4456, 4489, 4506, 4716
\@@_rectanglecolor:nnn ... 4462, 4471, 4474
\@@_renew_NC@rewrite@S: .... 202, 204, 1236
\@@_renew_dots: ..... 1134, 1226
\l_@@_renew_dots_bool .... 607, 1226, 7319
\@@_renew_matrix: ..... 777, 6511, 7321
\l_@@_respect_arraystretch_bool .....
503, 625, 5711, 5791, 5802, 5901, 5918, 5985
\l_@@_respect_blocks_bool 4512, 4529, 4552
\@@_restore_iRow_jCol: ..... 3137, 3183
\c_@@_revtex_bool .... 55, 57, 60, 62, 1477
\l_@@_right_delim_dim .....
..... 1283, 1287, 1293, 2567, 2905
\g_@@_right_delim_tl .. 1292, 1482, 1640,
1646, 1717, 2065, 2094, 2095, 2116, 2121, 5114
\l_@@_right_margin_dim .....
.... 511, 616, 1536, 2906, 3661, 5111, 5643
\@@_rotate: ..... 1218, 4250
\g_@@_rotate_bool .....
.. 240, 946, 974, 1959, 2022, 2340, 2850,
2895, 4250, 5795, 5839, 5844, 5905, 5922, 6125
\@@_rotate_cell_box: .....
.... 934, 974, 2022, 2340, 2850, 2895, 6125
\l_@@_rounded_corners_dim .....
304, 5950, 6097, 6312, 6313, 6348, 6401, 6495
\@@_roundedrectanglecolor .... 1345, 4465
\l_@@_row_max_int .... 295, 3252, 3395, 3412
\l_@@_row_min_int .... 294, 3320, 3393, 3410
\g_@@_row_of_col_done_bool .....
..... 271, 1105, 1487, 2645
\g_@@_row_style_tl .....
..... 275, 881, 1510, 1951, 4357, 4358,
4360, 4363, 4374, 4385, 4390, 4401, 4402, 5789
\g_@@_row_total_int .... 225, 1233, 1313,
1319, 1392, 1588, 2409, 2516, 3013, 3020,
3451, 3473, 4165, 5511, 5525, 5552, 5647,
6022, 6155, 6173, 6714, 6875, 6889, 7174, 7514
\@@_rowcolor ..... 1346, 4348, 4438
\@@_rowcolor_tabular ..... 1147, 4710
\@@_rowcolors ..... 1347, 4590
\@@_rowcolors_i:nnnnn ..... 4556, 4592
\l_@@_rowcolors_restart_bool ... 4515, 4544
\@@_rowlistcolors ..... 1348, 4519, 4591
\g_@@_rows_seq . 2595, 2597, 2599, 2601, 2603
\l_@@_rows_tl .....
..... 4434, 4482, 4558, 4575, 4623, 4648
\l_@@_rule_width_dim ..... 4776,
4934, 5129, 5186, 5229, 5237, 5256, 5276, 5285
\l_@@_rules_color_tl .....
..... 266, 543, 1527, 1528, 6950, 6951
\@@_set_CT@arc@: ..... 179, 1528, 4769, 6951
\@@_set_CT@arc@i: ..... 180, 181
\@@_set_CT@arc@ii: ..... 180, 183
\@@_set_CT@drsc@: ..... 185, 4771
\@@_set_CT@drsc@i: ..... 186, 187
\@@_set_CT@drsc@ii: ..... 186, 189
\@@_set_final_coords: ..... 3421, 3446
\@@_set_final_coords_from_anchor:n ..
... 3437, 3526, 3557, 3638, 3647, 3713, 3761
\@@_set_initial_coords: ..... 3416, 3435
\@@_set_initial_coords_from_anchor:n .
... 3426, 3519, 3554, 3637, 3644, 3707, 3755
\@@_set_size:n ..... 6529, 6544
\c_@@_siunitx_loaded_bool ... 192, 196, 201
\c_@@_size_seq .....
... 1256, 1261, 1311, 1312, 1313, 1314, 3016
\l_@@_small_bool ..... 775, 815, 821,
843, 878, 1160, 2059, 2088, 2832, 2878, 3089
\@@_standard_cline ..... 136, 1205
\@@_standard_cline:w ..... 136, 137
\l_@@_standard_cline_bool .. 477, 555, 1204
\c_@@_standard_tl ..... 490, 491, 3788
\l_@@_start_int ..... 4756, 4794, 4975
\g_@@_static_num_of_col_int .....
.... 299, 1657, 1705, 6010, 7419, 7438, 7654
\l_@@_stop_loop_bool ..... 3246, 3247,
3279, 3292, 3301, 3314, 3315, 3347, 3360, 3369
\@@_store_in_tmpb_tl ..... 6290, 6292
\@@_stroke_block:nnn ..... 6069, 6294
\@@_stroke_borders_block:nnn ... 6081, 6397
\@@_stroke_borders_block_i: .... 6414, 6419
\@@_stroke_borders_block_ii: ... 6422, 6426
\@@_stroke_horizontal:n .. 6437, 6439, 6472
\@@_stroke_vertical:n .... 6433, 6435, 6454

```

$\backslash@@_sub_matrix:nnnnnn$ 6844, 6870
 $\backslash@@_sub_matrix_i:nnnn$ 6911, 6917
 $\backslash l_@@_submatrix_extra_height_dim$
..... 316, 6760, 6945
 $\backslash l_@@_submatrix_hlines_clist$
..... 321, 6772, 6792, 6984, 6986
 $\backslash l_@@_submatrix_left_xshift_dim$
..... 317, 6762, 6997, 7030
 $\backslash l_@@_submatrix_name_str$
6807, 6879, 7019, 7021, 7033, 7035, 7043, 7047
 $\backslash g_@@_submatrix_names_seq$
..... 290, 3120, 6804, 6808, 7595
 $\backslash l_@@_submatrix_right_xshift_dim$
..... 318, 6764, 7006, 7040
 $\backslash g_@@_submatrix_seq$... 298, 1274, 3397, 6830
 $\backslash l_@@_submatrix_slim_bool$ 6770, 6887, 7535
 $\backslash l_@@_submatrix_vlines_clist$
..... 322, 6774, 6794, 6967, 6969
 $\backslash l_@@_suffix_tl$ 5579, 5590,
5600, 5603, 5652, 5660, 5661, 5679, 5687, 5688
 $\backslash l_@@_tabular_width_dim$
..... 238, 1066, 1068, 1744, 2999
 $\backslash l_@@_tabularnote_tl$ 361, 830, 858, 2438, 2447
 $\backslash g_@@_tabularnotes_seq$
..... 360, 400, 2453, 2459, 2475
 $\backslash c_@@_tabularx_loaded_bool$... 24, 42, 2986
 $\backslash@@_test_hline_in_block:nnnnn$
..... 4980, 4982, 5290
 $\backslash@@_test_hline_in_stroken_block:nnnn$.
..... 4984, 5312
 $\backslash@@_test_if_cell_in_a_block:nn$
..... 5368, 5386, 5404, 5424
 $\backslash@@_test_if_cell_in_block:nnnnnnn$...
..... 5430, 5432
 $\backslash@@_test_if_math_mode:$ 243, 1492, 2928
 $\backslash@@_test_in_corner_h:$ 4985, 5006
 $\backslash@@_test_in_corner_v:$ 4804, 4825
 $\backslash@@_test_vline_in_block:nnnnn$
..... 4799, 4801, 5301
 $\backslash@@_test_vline_in_stroken_block:nnnn$.
..... 4803, 5323
 $\backslash l_@@_the_array_box$
1280, 1295, 1554, 1562, 2426, 2427, 2429, 2432
 $\backslash c_@@_tikz_loaded_bool$
..... 25, 46, 1335, 3100, 5942, 7622
 $\backslash l_@@_tikz_rule_tl$
... 4773, 4857, 4937, 4938, 5038, 5132, 5133
 $\backslash l_@@_tikz_seq$ 302, 5943, 6100, 6109
 $\backslash l_@@_tmpc_dim$ 280, 1447, 1454,
3209, 3213, 4644, 4645, 4668, 4872, 4883,
4891, 4896, 4902, 4936, 4940, 5053, 5070,
5075, 5081, 5131, 5135, 6134, 6139, 6194,
6323, 6334, 6459, 6464, 6469, 6639, 6642, 6650
 $\backslash l_@@_tmpc_int$
4547, 4548, 4549, 5231, 5232, 5250, 5278, 5279
 $\backslash l_@@_tmpc_tl$... 4327, 4335, 4340, 4349,
4477, 4479, 4482, 4641, 4660, 6356, 6368,
6379, 6384, 6409, 6439, 6456, 6825, 6827, 6831
 $\backslash l_@@_tmpd_dim$ 281,
1449, 1455, 3211, 3214, 4665, 4668, 4884,
4891, 5063, 5070, 6136, 6139, 6172, 6183,
6187, 6190, 6194, 6332, 6335, 6641, 6642, 6660
 $\backslash l_@@_tmpd_tl$ 4478, 4480, 4483, 6357, 6361,
6380, 6391, 6410, 6435, 6474, 6826, 6828, 6831
 $\backslash g_@@_total_X_weight_int$
..... 267, 1153, 1541, 1544, 1563, 2161
 $\backslash l_@@_type_of_col_tl$ 813,
814, 2957, 2959, 6536, 6537, 6538, 6546, 6551
 $\backslash c_@@_types_of_matrix_seq$
..... 7369, 7370, 7375, 7379
 $\backslash@@_underbrace_i:n$ 7232, 7270
 $\backslash@@_update_for_first_and_last_row:$..
..... 917, 982, 1265, 2852, 2897
 $\backslash@@_use_arraybox_with_notes:$... 1602, 2494
 $\backslash@@_use_arraybox_with_notes_b:$. 1599, 2478
 $\backslash@@_use_arraybox_with_notes_c:$
..... 1600, 1631, 2422, 2492, 2531
 $\backslash l_@@_v_center_bool$.. 5987, 6031, 6036, 6211
 $\backslash c_@@_varwidth_loaded_bool$... 26, 30, 2000
 $\backslash@@_vdottedline:n$ 2193, 5458
 $\backslash@@_vline:n$ 1851, 4779, 4958, 5288, 5459, 6365
 $\backslash@@_vline_i:$ 4786, 4789
 $\backslash@@_vline_ii:$ 4814, 4822, 4850
 $\backslash@@_vline_iii:$ 4858, 4862
 $\backslash@@_vline_iv:$ 4855, 4910
 $\backslash@@_vline_v:$ 4859, 4926
 $\backslash@@_vlines_block:nnn$ 6043, 6351
 $\backslash l_@@_vlines_block_bool$ 313, 5957, 6039, 6060
 $\backslash l_@@_vlines_clist$ 320, 585, 597,
602, 638, 1155, 1690, 1696, 1727, 1739,
2201, 2208, 2352, 2359, 2755, 3099, 4956, 4957
 $\backslash l_@@_vpos_col_str$
1873, 1876, 1878, 1883, 1905, 1921, 1997, 2150
 $\backslash l_@@_vpos_of_block_tl$.. 310, 311, 5705,
5707, 5805, 5819, 5830, 5909, 5927, 5978, 5980
 $\backslash@@_w:$ 1681, 1766, 2280
 $\backslash l_@@_weight_int$
2146, 2151, 2152, 2155, 2158, 2159, 2161, 2165
 $\backslash l_@@_width_dim$ 233,
770, 851, 1554, 1562, 2967, 2968, 2989, 2990
 $\backslash g_@@_width_first_col_dim$
..... 284, 1486, 1593, 2640, 2853, 2854
 $\backslash g_@@_width_last_col_dim$
..... 283, 1485, 1652, 2798, 2898, 2899
 $\backslash l_@@_width_used_bool$ 291, 852, 1539
 $\backslash l_@@_x_final_dim$
..... 278, 3423, 3472, 3481, 3482, 3485,
3488, 3489, 3640, 3656, 3664, 3668, 3672,
3674, 3679, 3681, 3711, 3720, 3728, 3768,
3776, 3817, 3832, 3841, 3875, 3927, 3943,
4302, 4917, 5108, 5111, 5113, 5115, 6886,
6902, 6903, 6909, 7006, 7023, 7040, 7211,
7218, 7219, 7225, 7228, 7244, 7261, 7276, 7291
 $\backslash l_@@_x_initial_dim$... 276, 3418, 3450,
3459, 3460, 3463, 3466, 3467, 3640, 3655,
3656, 3663, 3668, 3672, 3674, 3676, 3679,
3681, 3720, 3728, 3768, 3776, 3814, 3831,
3841, 3875, 3927, 3941, 3943, 3961, 3963,
4299, 4916, 5098, 5101, 5103, 5105, 6885,
6895, 6896, 6906, 6997, 7022, 7030, 7190,
7197, 7198, 7204, 7207, 7244, 7261, 7276, 7291
 $\backslash l_@@_xdots_color_tl$ 516, 530, 3506, 3545,
3626, 3627, 3695, 3743, 3797, 4169, 4244, 4261
 $\backslash l_@@_xdots_down_tl$... 536, 3804, 3825, 3860

<code>\l_@@_xdots_line_style_tl</code>	489, 491, 526, 3788, 3797	<code>\AutoNiceMatrixWithDelims</code> ...	6540, 6580, 6592
<code>\l_@@_xdots_shorten_dim</code>	483, 485, 534, 3093, 3811, 3812, 3901, 3912, 3920	B	
<code>\l_@@_xdots_up_tl</code>	537, 3803, 3824, 3850	<code>\baselineskip</code>	115, 121, 1980
<code>\l_@@_y_final_dim</code>	279, 3424, 3524, 3528, 3566, 3570, 3572, 3597, 3607, 3608, 3722, 3725, 3770, 3773, 3817, 3832, 3840, 3877, 3932, 3951, 4303, 4921, 5096, 6710, 6732, 6747, 6923, 6938, 6939, 6944, 6962, 6979, 7023, 7031, 7041	<code>\begingroup</code>	2225
<code>\l_@@_y_initial_dim</code>	277, 3419, 3517, 3527, 3565, 3566, 3570, 3572, 3579, 3589, 3590, 3722, 3727, 3770, 3775, 3814, 3831, 3840, 3877, 3932, 3949, 3951, 3961, 3964, 4300, 4919, 5095, 6708, 6732, 6747, 6920, 6931, 6932, 6944, 6961, 6978, 7022, 7031, 7041	<code>\bfseries</code>	4394, 4397
<code>\</code> ..	2586, 2608, 6558, 6564, 6570, 6688, 7306, 7307, 7336, 7345, 7426, 7438, 7443, 7448, 7453, 7458, 7496, 7501, 7507, 7514, 7520, 7521, 7536, 7541, 7555, 7560, 7571, 7577, 7583, 7589, 7590, 7601, 7613, 7623, 7635, 7642, 7649, 7658, 7664, 7672, 7679, 7686, 7692, 7698, 7729, 7733, 7738, 7744, 7750, 7761, 7767, 7768, 7778, 7779, 7793, 7794, 7810, 7814, 7815, 7833, 7834, 7850, 7851, 7895, 7896, 7948, 7949, 8000, 8001, 8056, 8057	<code>\bgroup</code>	1480
<code>\{</code>	259, 1771, 2079, 2104, 2935, 6587, 7002, 7582, 7679, 7895, 8000	<code>\Block</code>	1217, 7619, 7629, 7684, 7711, 7750
<code>\}</code>	259, 1774, 2079, 2089, 2935, 6587, 7011, 7582, 7679, 7895, 8000	<code>\BNiceMatrix</code>	6526
<code>\ </code>	2937, 6586	<code>\bNiceMatrix</code>	6523
<code>\</code> ..	7393, 7403, 7410, 7418, 7419, 7437, 7438, 7513, 7514, 7527, 7533, 7537, 7545, 7576, 7588, 7594, 7606, 7653, 7654, 7655, 7663, 7669, 7677, 7684, 7697, 7723, 7724, 7732, 7767	<code>\Body</code>	1303
A		<code>\boldmath</code>	4394, 4397
<code>\A</code>	6802	bool commands:	
<code>\aboverulesep</code>	2469	<code>\bool_do_until:Nn</code>	3247, 3315
<code>\addtocounter</code>	417	<code>\bool_gset_false:N</code>	946, 993, 994, 1104, 1237, 1357, 1487, 1511, 1689, 2864, 2911, 5299, 5310, 5321, 5332, 5844
<code>\alph</code>	363	<code>\bool_gset_true:N</code>	1514, 1858, 2645, 2828, 2873, 2964, 3986, 4002, 4018, 4042, 4065, 4076, 4250, 4797, 4978
<code>\anchor</code>	3195, 3196	<code>\bool_if:NnTF</code>	178, 701, 710, 875, 878, 974, 1077, 1105, 1156, 1160, 1165, 1225, 1226, 1255, 1260, 1275, 1332, 1335, 1359, 1362, 1364, 1384, 1477, 1479, 1493, 1503, 1539, 1572, 1650, 1655, 1674, 1679, 1707, 1719, 1959, 2022, 2059, 2088, 2340, 2465, 2631, 2648, 2666, 2684, 2697, 2723, 2760, 2794, 2799, 2832, 2850, 2878, 2895, 2986, 3008, 3010, 3012, 3074, 3089, 3100, 3569, 3571, 3714, 3762, 3984, 4000, 4016, 4040, 4063, 4388, 4529, 4552, 5102, 5112, 5191, 5198, 5200, 5227, 5274, 5377, 5395, 5413, 5475, 5485, 5507, 5791, 5795, 5839, 5901, 5905, 5918, 5922, 6031, 6039, 6049, 6125, 6152, 6196, 6223, 6260, 6599, 7188, 7209, 7348, 7358, 7384, 7535
<code>\array</code>	1466	<code>\bool_if:nTF</code>	201, 369, 396, 1052, 1582, 3402, 4275, 4607, 4947, 4951, 5142, 5146, 5501, 5752, 6059, 6210, 6250, 6711, 6721, 6723, 6736, 6742, 6751, 7622
<code>\arraybackslash</code>	1952, 2176, 2314	<code>\bool_lazy_all:nTF</code>	1723, 1735, 2751, 4873, 5054, 5292, 5303, 5314, 5325, 5798
<code>\arraycolor</code>	1349	<code>\bool_lazy_and:nnTF</code>	2424, 2687, 2765, 2771, 2779, 2833, 3558, 3823, 4082, 4618, 5203, 6316, 6971, 6988
<code>\arraycolsep</code>	615, 617, 619, 1065, 1163, 1286, 1287, 1630, 1634, 2427, 2766, 2772, 2782, 5103, 5113	<code>\bool_lazy_or:nnTF</code>	523, 986, 1044, 1715, 2407, 2436, 2514, 2881, 3635, 3787, 3881, 4599, 4631, 4635, 4685, 4689, 5369, 5387, 5405, 5727, 5732, 6874, 7173, 7202, 7223
<code>\arrayrulecolor</code>	112	<code>\bool_lazy_or_p:nn</code>	2836
<code>\arrayrulewidth</code>	144, 149, 169, 545, 908, 1081, 1084, 1090, 1121, 1625, 1636, 1693, 1699, 1788, 1843, 2204, 2211, 2355, 2362, 2486, 2525, 2652, 2654, 2660, 2670, 2672, 2678, 2701, 2703, 2709, 2727, 2730, 2736, 2764, 2770, 2782, 2785, 4644, 4645, 4647, 4663, 4665, 4882, 4883, 4886, 4899, 4905, 5065, 5078, 5084, 5168, 5231, 5278, 5492, 6298, 6353, 6376, 6399, 6732, 6945, 6949	<code>\bool_not_p:n</code>	1726, 1728, 1738, 1740, 2689, 2754, 2756, 7203, 7224
<code>\arraystretch</code>	1162, 3580, 3598, 5792, 5902, 5919, 6921, 6924	<code>\bool_set:Nn</code>	3639
<code>\AutoNiceMatrix</code>	6588	<code>\c_false_bool</code>	2099, 2107, 2120, 2126, 2132, 3980, 3996, 4012
		<code>\g_tmpa_bool</code>	4797, 4805, 4832, 4840, 4845, 4978, 4986, 5013, 5021, 5026, 5299, 5310, 5321, 5332
		<code>\g_tmpb_bool</code>	1689, 1719, 1858
		<code>\l_tmpb_bool</code>	5189, 5198, 5257, 5374, 5388, 5406, 5428, 5441
		<code>\c_true_bool</code>	2078
		box commands:	
		<code>\box_clear_new:N</code>	1158, 1280

`\box_dp:N` 902, 922, 959, 979, 1034,
 1186, 1195, 1270, 2319, 2376, 3598, 5874, 6924
`\box_gclear_new:N` 5781
`\box_grotate:Nn` 5841
`\box_ht:N` 903, 924, 930, 942, 965,
 977, 1026, 1188, 1190, 1193, 1268, 1947,
 1971, 1973, 1979, 2315, 2375, 3580, 5865, 6921
`\box_ht_plus_dp:N` 2544, 2557
`\box_move_down:nn` 1034, 1977
`\box_move_up:nn`
 81, 83, 85, 1026, 2419, 2492, 2531
`\box_rotate:Nn` 936
`\box_set_dp:Nn` 958, 978, 2376
`\box_set_ht:Nn` 964, 976, 2375
`\box_set_wd:Nn` 952, 2426
`\box_use:N` 420, 943, 1033, 1982
`\box_use_drop:N` 984, 990, 1009, 2024, 2342,
 2378, 2419, 2420, 2432, 2859, 5884, 6245, 6282
`\box_wd:N` 421, 953, 981, 988,
 1047, 1291, 1293, 1554, 1562, 2427, 2429,
 2551, 2563, 2854, 2857, 2899, 2903, 5853, 6606
`\l_tmpa_box`
 .. 403, 420, 421, 1290, 1291, 1292, 1293,
 1617, 2375, 2376, 2378, 2419, 2420, 2544, 2557
`\l_tmpb_box` 2537, 2551, 2552, 2563

C

`\c` 64, 1711
`\Cdots` 1208, 3991, 3994
`\cdots` 1137, 1200
`\cellcolor` 1146, 1343
`\centering` 1916, 5809
char commands:
`\char_set_catcode_space:n` 1667
`\chessboardcolors` 1351
`\cline` 172, 1205, 1206
clist commands:
`\clist_clear:N` 344, 4678
`\clist_if_empty:Ntf` 4804, 4985, 6077
`\clist_if_empty_p:N` 2755
`\clist_if_in:NnTF` 342, 1113, 1696, 2208,
 2359, 4957, 5152, 6432, 6434, 6436, 6438, 6475
`\clist_map_inline:Nn`
 345, 4625, 4648, 4679, 5337, 6969, 6986
`\clist_map_inline:nn` . 2952, 4488, 4537, 6502
`\clist_new:N` 303, 319, 320, 321, 322, 498
`\clist_put_right:Nn` 354, 4696
`\clist_set:Nn` 597, 598, 602, 603, 637, 638, 639
`\clist_set_eq:NN` 4677
`\l_tmpa_clist` 344, 354, 356, 4677, 4679
`\CodeAfter`
 870, 1221, 2589, 2592, 2827, 2872, 3119, 7781
`\CodeBefore` 1475
`\color` 116, 122, 182, 184, 188, 190,
 880, 1621, 1639, 3500, 3503, 3506, 3539,
 3542, 3545, 3620, 3623, 3627, 3695, 3743,
 4163, 4166, 4169, 4238, 4241, 4244, 4261,
 4386, 4425, 5788, 5952, 6741, 7104, 7128, 7184
`\colorlet` ... 251, 252, 887, 894, 1152, 2842, 2886
`\columncolor` 1148, 1350, 3128, 4728
`\cr` 148, 174, 2821, 7258, 7262, 7292, 7294
`\crr` 2625, 7257, 7289

cs commands:

`\cs_generate_variant:Nn`
 63, 176, 3820, 4417, 4418, 5497, 5498
`\cs_gset:Npn` 116, 122, 5490
`\cs_gset_eq:NN` 74, 1179, 1495, 3138
`\cs_if_exist:Ntf`
 62, 1248, 1250, 1399, 1496, 1499,
 1683, 2040, 3185, 3186, 3282, 3295, 3350,
 3363, 3453, 3475, 3583, 3601, 4131, 4149,
 4206, 4224, 5193, 5477, 5530, 6157, 6175,
 6444, 6716, 6891, 6898, 6927, 6934, 7193, 7214
`\cs_if_exist_p:N`
 524, 4876, 5057, 5371, 5390, 5408
`\cs_if_free:Ntf`
 70, 2977, 3495, 3534, 3615, 3690, 3738
`\cs_if_free_p:N` 4277, 4279
`\cs_new_protected:Npx` 3165, 4288, 6419
`\cs_set:Nn` 689, 692, 695
`\cs_set:Npn` 112, 113, 118, 119,
 124, 136, 137, 151, 153, 154, 160, 162, 182,
 184, 188, 190, 366, 1129, 1471, 1472, 2226,
 2257, 2979, 3241, 3303, 3371, 4173, 4248,
 5156, 5157, 5163, 5164, 5262, 5792, 5902, 5919
`\cs_set_nopar:Npn` 1067, 1162, 1173, 5672, 5673
`\cs_set_nopar:Npx` 1068
`\cs_set_protected:Npn` 6577
`\cs_set_protected_nopar:Npn` 5769, 6112

D

`\Ddots` 1210, 4024, 4025, 4030, 4031
`\ddots` 1139, 1202
`\diagbox` 1222, 5769, 6112
dim commands:
`\dim_add:Nn` 5641
`\dim_compare:nNnTF` .. 115, 121, 950, 956,
 962, 1744, 2685, 2903, 2967, 3463, 3485,
 3674, 4361, 4372, 5554, 5564, 6167, 6187, 6606
`\dim_compare_p:n` 5801
`\dim_gzero:N` 954, 960, 966
`\dim_max:nn` 5543, 5547
`\dim_min:nn` 5535, 5539
`\dim_ratio:nn` 3729, 3777,
 3895, 3900, 3911, 3919, 3928, 3933, 3944, 3952
`\dim_set:Nn` 5516, 5523, 5534, 5538,
 5542, 5546, 5558, 5559, 5568, 5569, 5613, 5626
`\dim_set_eq:NN` 5514, 5521, 5621, 5635
`\dim_sub:Nn` 5638
`\dim_use:N`
 5555, 5565, 5616, 5617, 5629, 5630,
 5653, 5654, 5655, 5656, 5680, 5681, 5682, 5683
`\dim_zero_new:N` 5513, 5515, 5520, 5522
`\c_max_dim` 3450, 3463,
 3472, 3485, 5514, 5516, 5521, 5523, 5555,
 5565, 6154, 6167, 6172, 6187, 6712, 6713,
 6885, 6886, 6906, 6909, 7190, 7204, 7211, 7225
`\dotfill` 1220, 6595
`\dots` 1141
`\doublerulesep`
 1844, 4886, 4900, 5065, 5079, 5169, 5232, 5279
`\doublerulesepcolor` 118
`\downbracefill` 7262
`\draw` 3808, 4938, 5133, 6468, 6488

E		<code>\futurelet</code> 131
<code>\egroup</code>	1472, 1665	
<code>\else</code>	2226	
else commands:		G
<code>\else:</code>	245, 4396	<code>\globaldefs</code> 634, 1661, 6015
<code>\endarray</code>	1470, 1472, 2577, 2605	group commands:
<code>\endBNiceMatrix</code>	6527	<code>\group_insert_after:N</code> 6600, 6601, 6603, 6604
<code>\endbNiceMatrix</code>	6524	
<code>\endgroup</code>	2234	H
<code>\endNiceArray</code>	2974, 2995, 3004	<code>\halign</code> 1197, 7255, 7287
<code>\endNiceArrayWithDelims</code>	2921, 2931	<code>\hbox</code> 1075, 1626, 2430,
<code>\endpgfscope</code>	3865, 6659	2492, 2531, 2650, 2668, 2695, 2699, 2725, 2762
<code>\endpNiceMatrix</code>	6515	hbox commands:
<code>\endsavenotes</code>	1674	<code>\hbox:n</code> 81, 83, 86
<code>\endtabular</code>	1472	<code>\hbox_gset:Nn</code> 5783
<code>\endtabularnotes</code>	2460	<code>\hbox_overlap_left:n</code> . 1027, 1035, 2629, 2855
<code>\endVNiceMatrix</code>	6521	<code>\hbox_overlap_right:n</code> 420, 2796, 2901
<code>\endvNiceMatrix</code>	6518	<code>\hbox_set:Nn</code> 403, 1024,
<code>\enskip</code>	2069, 2097, 2105, 2118, 2124	1290, 1292, 1617, 1975, 2175, 2537, 2552, 6124
<code>\ensuremath</code>	3985, 4001, 4017, 4041, 4064	<code>\hbox_set:Nw</code> 874, 1295, 2013, 2331, 2830, 2876
<code>\everycr</code>	147, 174, 1183, 7254, 7284	<code>\hbox_set_end:</code>
<code>\everypar</code>	1945, 1948	972, 1538, 2021, 2339, 2849, 2894
exp commands:		<code>\hbox_to_wd:nn</code> 445, 470, 7260, 7290
<code>\exp_after:wN</code>		<code>\Hdotsfor</code> 1215, 7393, 7606
.	208, 1528, 1622, 1640, 1704, 2231, 6951	<code>\hdotsfor</code> 1142
<code>\exp_args:NNc</code>	5479	<code>\hdottedline</code> 1212
<code>\exp_args:Nne</code>	2959	<code>\heavyrulewidth</code> 2470
<code>\exp_args:NNV</code>	2597, 3972,	<code>\hfil</code> 1767, 2281, 7257, 7289
.	3988, 4004, 4020, 4044, 4102, 4179, 4257, 6840	<code>\hfill</code> 144, 169
<code>\exp_args:NnV</code>	5212, 5235	<code>\Hline</code> 1213, 5159
<code>\exp_args:NNx</code>	1112, 2207, 2358	<code>\hline</code> 124
<code>\exp_args:Nnx</code>	2157, 5216	hook commands:
<code>\exp_args:No</code>	3796	<code>\hook_gput_code:nnn</code>
<code>\exp_args:NV</code> 27, 90, 106, 193, 199, 367,
.	1684, 2232, 2573, 2600, 2602, 4937, 5132, 6307	481, 484, 487, 500, 533, 699, 708, 1228,
<code>\exp_args:Nxx</code>	5762, 5763	3163, 3968, 4098, 4175, 4253, 4286, 6417, 6836
<code>\exp_last_unbraced:NV</code>	1360, 3121, 6089	<code>\hrule</code> 128, 144, 169, 1121, 2470, 6746, 7109, 7133
<code>\exp_not:N</code>	47, 48, 51, 52,	<code>\hsize</code> 1959
.	1788, 1837, 1909, 1911, 1916, 1917, 1918,	<code>\hskip</code> 127
.	3016, 3030, 3038, 3040, 3131, 4365, 4376,	<code>\Hspace</code> 1214
.	4386, 4728, 4938, 5133, 5284, 5356, 5819,	<code>\hspace</code> 4077
.	5830, 5909, 5927, 6092, 6468, 6488, 6550, 7057	<code>\hss</code> 1767, 2281
<code>\exp_not:n</code>	1058, 1671,	I
.	3132, 4112, 4113, 4189, 4705, 4716, 4718,	<code>\ialign</code> 1072, 1173, 1196, 3112, 5994
.	4729, 5219, 5778, 5882, 5894, 5895, 6044,	<code>\Iddots</code> 1211, 4048, 4049, 4054, 4055
.	6054, 6070, 6082, 6094, 6121, 6552, 6616, 6617	<code>\iddots</code> 76, 1140, 1203
<code>\expandafter</code>	2978	if commands:
<code>\ExplSyntaxOff</code>	72, 1673, 5494	<code>\if_mode_math:</code> 245, 4392
<code>\ExplSyntaxOn</code>	69, 1666, 5487	<code>\IfBooleanTF</code> 1529
<code>\extrarowheight</code>	3580, 5793, 5903, 5920, 6921	<code>\ifnum</code> 126, 4357, 5156, 5179
F		<code>\ifstandalone</code> 1499
<code>\fi</code>	126, 1686, 2226, 2229, 4401, 5156, 5179	<code>\ignorespaces</code> 2264, 4403
fi commands:		int commands:
<code>\fi:</code>	247, 4398	<code>\int_case:nnTF</code> 4022, 4028, 4046, 4052
<code>\five</code>	3190, 3195	<code>\int_compare:nNnTF</code> 139, 140, 164, 872, 873,
flag commands:		884, 891, 927, 937, 1118, 1120, 1252, 1258,
<code>\flag_clear_new:n</code>	7050	1263, 1541, 1544, 1570, 1574, 1585, 1589,
<code>\flag_height:n</code>	7075	1590, 1657, 1970, 2155, 2235, 2262, 2448,
<code>\flag_raise:n</code>	7074	2487, 2526, 2598, 2626, 2747, 2879, 3257,
<code>\fontdimen</code>	2415	3264, 3268, 3270, 3325, 3332, 3336, 3338,
fp commands:		3502, 3541, 3622, 3657, 3659, 4165, 4240,
<code>\fp_eval:n</code>	3836	4344, 4594, 4639, 4657, 4693, 4724, 4741,
<code>\fp_to_dim:n</code>	3871	4742, 4749, 4750, 4785, 4807, 4811, 4819,

6216, 6248, 6252, 6325, 6327, 6554, 6556, 6558, 6562, 6564, 6566, 6568, 6570, 6955, 6957	
\int_compare_p:n	
3404, 3405, 3406, 3407, 4609, 4611, 6317, 6318	
\int_do_until:nNnn	4549
\int_gadd:Nn	2161, 2261
\int_gincr:N	
. 871, 900, 1502, 1817, 1934, 2027, 2052, 2184, 2721, 2750, 2874, 3716, 3764, 5472, 5768	
\int_if_even:nTF	4497, 7075
\int_incr:N	399, 1827, 4577
\int_min:nn	3204, 3206, 3208, 3210, 3220, 3222, 3385, 3412, 3413
\int_mod:nn	4565
\int_step_inline:nn ... 1415, 1421, 3046, 3052, 3060, 3066, 3202, 4493, 4495, 6968, 6985	
\int_step_inline:nnnn 5366, 5384, 5399, 5402	
\c_zero_int	2060, 4410, 4600
io commands:	
\iow_now:Nn ... 67, 93, 1666, 1667, 1668, 1673	
\iow_shipout:Nn	5487, 5488, 5494
\item	2453, 2459
K	
\kern	86
keys commands:	
\keys_define:nn	
..... 519, 541, 548, 628, 679, 717, 724, 768, 805, 819, 836, 849, 1367, 1863, 2145, 4068, 4306, 4510, 4752, 4762, 5181, 5216, 5242, 5461, 5691, 5939, 6342, 6441, 6492, 6534, 6672, 6677, 6758, 6779, 6788, 7155, 7317	
\l_keys_key_str	2146, 7306, 7479, 7485, 7599, 7618, 7738, 7749, 7766, 7778, 7793, 7814, 7832, 7849, 7894, 7947, 7999
\keys_set:nn	551, 553, 567, 794, 797, 804, 1371, 1379, 1524, 1525, 1897, 1999, 2046, 2154, 2958, 2970, 2991, 3000, 3141, 3505, 3544, 3625, 3694, 3742, 4168, 4243, 4260, 4334, 4526, 4853, 5034, 5474, 6030, 6405, 6679, 6683, 6813, 6880, 7183
\keys_set_known:nn	
4034, 4058, 5190, 5744, 6299, 6354, 6377, 6400	
\keys_set_known:nnN	
..... 1802, 2153, 4784, 4966, 6547	
\l_keys_value_tl 7519, 7637, 7644, 8051	
L	
\Ldots	1207, 3975, 3978
\ldots	1136, 1199
\leaders	144, 169
\left	1622, 2540, 2555, 6742, 7105, 7129
legacy commands:	
\legacy_if:nTF	663
\line	3116, 7582, 7789
\linewidth	2968
M	
\makebox	2024, 2342
\mathinner	78
mode commands:	
\mode_leave_vertical: 1491, 2313, 4386	
msg commands:	
\msg_error:nn	12
\msg_error:nnn	13
\msg_error:nnnn	14, 6012, 6019, 6023
\msg_fatal:nn	15
\msg_fatal:nnn	16
\msg_new:nnn	17
\msg_new:nnnn	18
\msg_redirect_name:nnn	20
\multicolumn . 1227, 1229, 4079, 4088, 4094, 4116	
\multispan	140, 141, 165, 166, 2224
\myfiledate	6
\myfileversion	7
N	
\newcolumntype	2987
\newcounter	359
\NewDocumentCommand ... 371, 394, 803, 1377, 3140, 3972, 3988, 4004, 4020, 4044, 4102, 4179, 4257, 4332, 4438, 4447, 4456, 4465, 4486, 4491, 4504, 4519, 4590, 4699, 4710, 4722, 6287, 6540, 6588, 6820, 6840, 7145, 7150	
\NewDocumentEnvironment	1474, 2570, 2579, 2914, 2924, 2954, 2965, 2984, 2996, 5470
\NewExpandableDocumentCommand	217, 5714
\newlist	375, 386
\NiceArray	2972, 2993, 3002
\NiceArrayWithDelims	2919, 2929
nicematrix commands:	
\g_nicematrix_code_after_tl	
..... 261, 672, 2594, 3121, 3123, 6041, 6051, 6067, 6079, 6691, 6698	
\g_nicematrix_code_before_tl 1244, 3125, 3132, 4337, 4346, 4703, 4714, 4726, 6090, 6102	
\NiceMatrixLastEnv	217
\NiceMatrixOptions	803, 7850, 8056
\NiceMatrixoptions	7641
\noalign	115, 121, 126, 149, 1100, 1179, 5156, 5264, 5448, 7259, 7293
\nobreak	389
\nointerlineskip	7259, 7293
\normalbaselines	1159
\NotEmpty	1223
\null	2260
\nulldelimiterspace 2551, 2563, 6727, 6947	
\nullfont 6738, 6745, 7101, 7108, 7125, 7132	
O	
\omit 139, 2628, 2644, 2720, 2746, 6687, 6688	
\OnlyMainNiceMatrix	1219, 4733
\OverBrace	3115, 7179, 7767
P	
\par	2447, 2455
\path	6504
peek commands:	
\peek_meaning:NTF . 180, 186, 401, 1130, 2980	
\peek_meaning_ignore_spaces:NTF 2572, 5159	
\peek_meaning_remove_ignore_spaces:NTF 172	
\peek_remove_spaces:n	
... 4701, 4712, 5716, 6822, 6842, 7147, 7152	
\pgfdeclareshape	3188
\pgfextracty	6219
\pgfgetlastxy	463
\pgfpathcircle	3960
\pgfpathlineto	4896, 4902, 5075, 5081, 6464, 6484, 6642, 6962, 6979, 7013

`\pgfpathmoveto` 4895, 4901,
 5074, 5080, 6463, 6483, 6637, 6961, 6978, 7004
`\pgfpathrectanglecorners` 4667, 4889, 5068, 6333
`\pgfpointadd` 461
`\pgfpointanchor` 191, 220, 3428,
 3439, 3456, 3478, 3586, 3604, 5533, 5541,
 6162, 6180, 6200, 6202, 6220, 6257, 6719,
 6894, 6901, 6930, 6937, 7049, 7053, 7196, 7217
`\pgfpointdiff` .. 462, 1323, 1329, 1396, 1410, 1411
`\pgfpointlineattime` 3830
`\pgfpointorigin` 2635, 2808, 3196
`\pgfpointscale` 461
`\pgfpointshapeborder` 4298, 4301
`\pgfrememberpicturepositiononpagetrue` ...
 905, 1000,
 1088, 1433, 2634, 2658, 2676, 2707, 2734,
 2776, 2805, 3174, 3201, 3785, 4297, 4865,
 4913, 4929, 5046, 5092, 5124, 5576, 5586,
 5597, 6127, 6301, 6428, 6632, 6705, 6882, 7186
`\pgfscope` 3827, 6649
`\pgfset` 430,
 455, 1001, 6234, 6271, 6648, 6726, 6884, 7230
`\pgfsetbaseline` 999
`\pgfsetcornersarced` 4666, 6309
`\pgfsetlinewidth` .. 4905, 5084, 6336, 6431, 6949
`\pgfsetrectcap` 4906, 5085
`\pgfsetroundcap` 6645
`\pgfsetstrokecolor` 6307
`\pgfsyspdfmark` 70, 71
`\pgftransformrotate` 3834
`\pgftransformshift` 436, 461, 3212, 3828, 6233,
 6255, 6650, 6660, 6728, 7027, 7037, 7241, 7273
`\pgfusepath` 3854, 3864, 4428, 4892, 6337
`\pgfusepathqfill` 3966, 5071
`\pgfusepathqstroke`
 4907, 5086, 6465, 6485, 6646, 6963, 6980, 7014
`\phantom` 3985, 4001, 4017, 4041, 4064
`\pNiceMatrix` 6514
prg commands:
`\prg_do_nothing:`
 74, 202, 538, 1179, 3119, 4759, 4760
`\prg_new_conditional:Nnn` 4597, 4605
`\prg_replicate:nn` 409, 2716,
 2717, 4116, 4897, 5076, 6557, 6560, 6563, 6569
`\prg_return_false:` 4602, 4614
`\prg_return_true:` 4603, 4613
`\ProcessKeysOptions` 7329
`\ProvideDocumentCommand` 76
`\ProvidesExplPackage` 4

Q

`\quad` 390
quark commands:
`\q_stop` 136, 137, 153, 154, 157, 158,
 160, 161, 162, 181, 183, 187, 189, 335, 348,
 1360, 1382, 1528, 1704, 1777, 1858, 2092,
 2114, 2231, 2282, 2589, 2592, 4251, 4265,
 4266, 4476, 4481, 4541, 4629, 4630, 4652,
 4653, 4683, 4684, 4769, 4771, 5670, 5677,
 5719, 5720, 5724, 6089, 6292, 6315, 6324,
 6355, 6358, 6378, 6381, 6408, 6411, 6529,
 6544, 6824, 6829, 6855, 6862, 6951, 7060, 7068

R

`\raggedleft` 1918, 5810, 6664
`\raggedright` 1917, 1959, 5811
`\rectanglecolor` 1344, 3127
`\refstepcounter` 418
regex commands:
`\regex_const:Nn` 64
`\regex_match:nnTF` 6802
`\regex_replace_all:NnN` 1709
`\renewcommand` 206
`\RenewDocumentEnvironment`
 6513, 6516, 6519, 6522, 6525
`\RequirePackage` 1, 3, 9, 10, 11
`\right` 1640, 2547, 2559, 6751, 7113, 7137
`\rotate` 1218
`\roundedrectanglecolor` 1345, 6092
`\rowcolor` 1147, 1346
`\rowcolors` 1347
`\rowlistcolors` 1348
`\RowStyle` 1224, 7833

S

`\savedanchor` 3190
`\savenotes` 1479
scan commands:
`\scan_stop:` 3122
`\scriptstyle`
 878, 2832, 2878, 3815, 3816, 3850, 3860
seq commands:
`\seq_clear:N` 1702
`\seq_clear_new:N` 4522, 5336
`\seq_count:N` 2599, 4413, 4567
`\seq_gclear:N` 1230, 1231, 1273,
 1274, 1276, 1505, 1506, 1507, 1508, 2475, 3120
`\seq_gclear_new:N` 1277, 1278, 1356, 2595, 2611
`\seq_gpop_left:NN` 2601, 2613
`\seq_gput_left:Nn` 668, 2237, 2239, 6062
`\seq_gput_right:Nn` 400, 1785, 2240,
 3382, 4412, 5879, 5891, 6074, 6619, 6808, 6830
`\seq_gset_from_clist:Nn`
 3016, 3030, 3038, 3040
`\seq_gset_map_x:NNn` 3144
`\seq_gset_split:Nnn` 2597, 2612
`\seq_if_empty:NTF` 2434, 3026, 3034, 5352, 6100
`\seq_if_empty_p:N` 4619
`\seq_if_in:NnTF`
 666, 4659, 4829, 4835, 4842, 5010,
 5016, 5023, 5536, 5544, 6160, 6178, 6804, 7379
`\seq_item:Nn`
 1256, 1261, 1311, 1312, 1313, 1314, 4586, 4588
`\seq_map_function:NN` 2603
`\seq_map_indexed_inline:Nn` 4408, 4423
`\seq_map_inline:Nn`
 1434, 2453, 2459, 2615, 3397, 4556, 4798,
 4800, 4802, 4979, 4981, 4983, 5429, 5995, 6953
`\seq_mapthread_function:NNN` 5665
`\seq_new:N` 234, 250, 285, 286, 287,
 288, 289, 290, 292, 293, 298, 302, 360, 7369
`\seq_put_right:Nn` 5416, 5943
`\seq_set_eq:NN` 4531
`\seq_set_filter:NNn` 4532, 4554
`\seq_set_from_clist:Nn` 5356, 7370
`\seq_set_map_x:NNn` 7375
`\seq_set_split:Nnn` 4523

<code>\xarraycr@array</code>	1463	<code>\tl_gclear:N</code>	869, 1510, 1517, 1688, 2230, 3118, 3123, 4427
<code>\xhline</code>	132	<code>\tl_gclear_new:N</code>	1238, 1239, 1240, 1241, 1242, 1243, 1244, 1509
<code>\array@array</code>	1466	<code>\tl_gput_left:Nn</code>	1052, 1721, 4726
<code>\bBigg@</code>	1290, 1292	<code>\tl_gput_right:Nn</code>	1052, 1546, 1733, 1787, 1835, 1849, 2077, 2098, 2106, 2119, 2125, 2131, 2192, 3014, 3028, 3036, 3129, 4104, 4181, 4337, 4346, 4358, 4363, 4374, 4385, 4415, 4703, 4714, 5171, 5267, 5282, 5287, 5354, 5453, 5771, 6041, 6051, 6067, 6079, 6090, 6102, 6114, 6609
<code>\c@MaxMatrixCols</code>	2946, 7412	<code>\tl_gset:Nn</code>	1481, 1482, 1483, 1670, 1692, 1698, 2064, 2065, 2095, 2121, 3131, 4413
<code>\c@tabularnote</code>	2437, 2448, 2476	<code>\tl_if_blank:nTF</code>	4440, 4443, 4449, 4452, 4458, 4461, 4467, 4470, 4574, 5718
<code>\col@sep</code>	1064, 1065, 1592, 1653, 2639, 2692, 2759, 2865, 2900, 3467, 3489	<code>\tl_if_blank_p:n</code>	4632, 4636, 4686, 4690, 4877, 5058, 5728, 5733
<code>\CT@arc</code>	112, 113	<code>\tl_if_empty:NnTF</code>	1108, 1518, 1527, 1620, 1638, 1803, 2447, 3098, 3099, 3125, 4335, 4383, 4571, 4654, 4655, 4857, 5038, 5195, 5786, 6065, 6087, 6303, 6461, 6481, 6740, 6950, 7103, 7127, 7184
<code>\CT@arc@</code>	111, 116, 130, 143, 168, 182, 184, 1495, 2470, 3138, 4904, 4922, 5083, 5117, 6306, 6430, 6644, 6952	<code>\tl_if_empty:nTF</code>	156, 646, 772, 807, 823, 839, 860, 1440, 2581, 2608, 3506, 3545, 3626, 3695, 3743, 4169, 4244, 4261, 6799, 7070, 7138, 7392
<code>\CT@drsc</code>	118, 119	<code>\tl_if_empty_p:N</code>	1727, 1739, 3824, 3825
<code>\CT@drsc@</code>	122, 188, 190, 4876, 4877, 4881, 5057, 5058, 5062	<code>\tl_if_empty_p:n</code>	2438, 5759
<code>\CT@everycr</code>	1177	<code>\tl_if_eq:NnTF</code>	1110, 1690, 2201, 2352, 2377, 2496, 4956, 5104, 5114, 5151, 6856, 6859, 6863, 6866, 6967, 6984
<code>\CT@row@color</code>	1179	<code>\tl_if_eq:nnTF</code>	658, 781, 2092, 2114, 4409
<code>\endarray@array</code>	1470	<code>\tl_if_eq_p:NN</code>	3788
<code>\if@firstamp</code>	2226	<code>\tl_if_exist:NnTF</code>	1512
<code>\if@tempswa</code>	1686, 2229	<code>\tl_if_in:NnTF</code>	4540, 4628, 4651, 4682, 5212
<code>\insert@column</code>	1461	<code>\tl_if_in:nnTF</code>	347, 2079, 2089, 2104
<code>\insert@column@array</code>	1461	<code>\tl_if_single_token:nTF</code>	590, 789
<code>\NC@</code>	1129, 2979	<code>\tl_if_single_token_p:n</code>	63
<code>\NC@do</code>	2978	<code>\tl_map_inline:nn</code>	2582
<code>\NC@find</code>	210	<code>\tl_new:N</code>	242, 249, 261, 262, 263, 266, 273, 275, 300, 301, 305, 310, 361, 489, 493, 514, 516, 517
<code>\NC@find@V</code>	1683	<code>\tl_put_left:Nn</code>	1157, 1365
<code>\NC@list</code>	1686, 2229, 2978	<code>\tl_put_right:Nn</code>	648, 1167, 1265, 1305, 1521
<code>\NC@rewrite@S</code>	206	<code>\tl_range:nnn</code>	101
<code>\new@ifnextchar</code>	1235	<code>\tl_set:Nn</code>	4482, 4483, 4542, 4558, 4561, 4638, 4640, 4656, 4658, 4692, 4694, 4793, 4974, 5745, 6326, 6328, 6359, 6360, 6382, 6383, 6412, 6413, 6857, 6860, 6864, 6867
<code>\newcol@</code>	1131, 1132, 2981, 2982	<code>\tl_set_eq:NN</code>	356, 491, 4479, 4480, 4641, 6356, 6357, 6379, 6380, 6409, 6410, 6827, 6828, 6858, 6861, 6865, 6868
<code>\nicematrix@redefine@check@rerun</code> ..	93, 96	<code>\tl_set_rescan:Nnn</code>	2596, 3971, 4101, 4178, 4256, 6839
<code>\pgf@relevantforpicturesizefalse</code> ..	1318, 1432, 3175, 3786, 3957, 4422, 4536, 4866, 4914, 4930, 5047, 5093, 5125, 5577, 5587, 5598, 6128, 6302, 6429, 6631, 6706, 6883, 7187	<code>\tl_to_str:n</code>	7376
<code>\pgfsys@getposition</code>	1316, 1321, 1327, 1394, 1402, 1405	token commands:	
<code>\pgfsys@markposition</code>	1029, 1037, 1082, 1169, 1315, 2632, 2653, 2671, 2702, 2728, 2767, 2801	<code>\token_to_str:N</code>	7393, 7490, 7495, 7501, 7519, 7527, 7533, 7537, 7582, 7588, 7606, 7619, 7628, 7677, 7684, 7697, 7711, 7733, 7749, 7766, 7767, 7780, 7789, 7795, 7833, 7850, 8056
<code>\pgfutil@check@rerun</code>	98, 99		
<code>\reserved@a</code>	131		
<code>\rvtx@iffformat@geq</code>	62		
<code>\set@color</code>	5787, 6124		
<code>\tikz@library@external@loaded</code>	1496		
tex commands:			
<code>\tex_mkern:D</code>	80, 82, 84, 87		
<code>\tex_the:D</code>	209		
<code>\textfont</code>	2415		
<code>\textit</code>	363		
<code>\textsuperscript</code>	364, 365		
<code>\the</code>	1686, 1704, 2229, 2231, 2978		
<code>\thetabularnote</code>	366		
<code>\tikzexternaldisable</code>	1498		
<code>\tikzset</code>	1337, 1500, 3102, 4937, 5132		
tl commands:			
<code>\tl_clear:N</code>	5187, 6297, 7182		
<code>\tl_clear_new:N</code>	4477, 4478, 4524, 4560, 6404, 6825, 6826, 6851, 6852, 6853, 6854		
<code>\tl_const:Nn</code>	47, 48, 51, 52, 490, 2823, 2868, 7061		
<code>\tl_count:n</code>	2396, 2508		

U		\vbox_top:n 7251	
\UnderBrace 3114, 7178, 7767		\vcenter 1623, 2541, 6743, 7106, 7130, 7733	
\unskip 2463		\Vdots 1209, 4007, 4010	
\upbracefill 7292		\vdots 1138, 1201	
use commands:		\Vdotsfor 1216	
\use:N 1055,		\vfill 444, 470	
1300, 1301, 1515, 1533, 1534, 2941, 2961, 4426		\vline 130	
\use:n		\VNiceMatrix 6520	
1902, 4262, 4733, 4938, 5133, 5817, 5828,		\vNiceMatrix 6517	
5907, 5925, 6363, 6386, 6468, 6488, 6548, 7056		\vrule 128, 1947, 2315, 2319	
\usepackage 7355, 7365		\vskip 127	
\usepgfmodule 2		\vtop 1079	
V		X	
vbox commands:		\xglobal 887, 894, 1152, 2842, 2886	
\vbox:n 86, 7285			
\vbox_set_top:Nn 939			
\vbox_to_ht:nn 441, 468, 2543, 2556		Z	
\vbox_to_zero:n 941		\Z 6802	

Contents

1	The environments of this package	2
2	The vertical space between the rows	2
3	The vertical position of the arrays	3
4	The blocks	4
4.1	General case	4
4.2	The mono-column blocks	6
4.3	The mono-row blocks	6
4.4	The mono-cell blocks	6
4.5	Horizontal position of the content of the block	7
5	The rules	7
5.1	Some differences with the classical environments	8
5.1.1	The vertical rules	8
5.1.2	The command <code>\cline</code>	8
5.2	The thickness and the color of the rules	9
5.3	The tools of nicematrix for the rules	9
5.3.1	The keys <code>hlines</code> and <code>vlines</code>	9
5.3.2	The keys <code>hvlines</code> and <code>hvlines-except-borders</code>	10
5.3.3	The (empty) corners	10
5.4	The command <code>\diagbox</code>	11
5.5	Dotted rules	11
5.6	Commands for customized rules	12
6	The color of the rows and columns	13
6.1	Use of <code>colortbl</code>	13
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	14
6.3	Color tools with the syntax of <code>colortbl</code>	18
7	The command <code>\RowStyle</code>	18

8	The width of the columns	19
8.1	Basic tools	19
8.2	The columns V of varwidth	20
8.3	The columns X	21
9	The exterior rows and columns	21
10	The continuous dotted lines	23
10.1	The option nullify-dots	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>	28
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>	35
13	Other features	36
13.1	Use of the column type S of siunitx	36
13.2	Alignment option in <code>{NiceMatrix}</code>	36
13.3	The command <code>\rotate</code>	36
13.4	The option <code>small</code>	37
13.5	The counters <code>iRow</code> and <code>jCol</code>	37
13.6	The option <code>light-syntax</code>	38
13.7	Color of the delimiters	38
13.8	The environment <code>{NiceArrayWithDelims}</code>	38
13.9	The command <code>\OnlyMainNiceMatrix</code>	39
14	Use of Tikz with <code>nicematrix</code>	39
14.1	The nodes corresponding to the contents of the cells	39
14.1.1	The columns V of varwidth	40
14.2	The medium nodes and the large nodes	40
14.3	The nodes which indicate the position of the rules	42
14.4	The nodes corresponding to the command <code>\SubMatrix</code>	43
15	API for the developpers	43
16	Technical remarks	44
16.1	Diagonal lines	44
16.2	The empty cells	45
16.3	The option <code>exterior-arraycolsep</code>	45
16.4	Incompatibilities	46

17	Examples	46
17.1	Utilisation of the key 'tikz' of the command <code>\Block</code>	46
17.2	Notes in the tabulars	47
17.3	Dotted lines	48
17.4	Dotted lines which are no longer dotted	49
17.5	Dashed rules	50
17.6	Stacks of matrices	50
17.7	How to highlight cells of a matrix	54
17.8	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	56
18	Implementation	57
19	History	236
	Index	244