

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

March 11, 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 6.8 of `nicematrix`, at the date of 2022/03/11.

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

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

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

1 The environments of this package

The package `nicematrix` defines the following new environments.

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

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

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

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

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

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

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

2 The vertical space between the rows

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

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

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

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

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

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

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

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

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

3 The vertical position of the arrays

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

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

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

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

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

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

1. an item

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

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

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

1. an item

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

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

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

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

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

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

4 The blocks

4.1 General case

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

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

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

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

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

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

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

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

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. 47;
- the key `name` provides a name to the rectangular Tikz node corresponding to the block; it's possible to use that name with Tikz in the `\CodeAfter` of the environment (cf. p. 28);
- **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).

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

4.3 The mono-row blocks

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

4.4 The mono-cell blocks

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

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

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

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

```

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

```

Writer	year of birth
Hugo	1802
Balzac	1799

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

4.5 Horizontal position of the content of the block

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

```

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

```

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

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

5 The rules

The usual techniques for the rules may be used in the environments of `nicematrix` (excepted `\vline`). However, there is some small differences with the classical environments.

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

5.1 Some differences with the classical environments

5.1.1 The vertical rules

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

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

First	Second
Peter	
Mary	George

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

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

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

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

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

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

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

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

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

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

5.5 Dotted rules

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

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

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[l]}[hvlines]
a rather long text which fits on several lines
& a rather long text which fits on several lines \\
a shorter text & a shorter text
\end{NiceTabular}
```

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

9 The exterior rows and columns

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

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

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

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

```

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

```

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

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

We have several remarks to do.

- For the environments with an explicit preamble (i.e. `{NiceTabular}`, `{NiceArray}` and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type `r` for the first column and `l` for the last one.²⁹
- One may wonder how `nicematrix` determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
 - For the environments with explicit preamble, like `{NiceTabular}` and `{pNiceArray}`, the number of columns can obviously be computed from the preamble.
 - When the option `light-syntax` (cf. p. 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}

```

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

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

```

\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 `\` 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}
```

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

10.1 The option `nullify-dots`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

10.3 How to generate the continuous dotted lines transparently

Imagine you have a document with a great number of mathematical matrices with ellipsis. You may wish to use the dotted lines of `nicematrix` without having to modify the code of each matrix. It's possible with the keys `renew-dots` and `renew-matrix`.³⁵

- The option `renew-dots`

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

- The option `renew-matrix`

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

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

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

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

10.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `\CodeAfter` which is described p. 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 paths (with the exception of “`color`”, “`shorten >`” and “`shorten <`”).

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

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

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.

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

10.6 The dotted lines and the rules

The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier `|` in the preamble, by the command `\Hline`, 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}
```

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

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}
```

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

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. 45).

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

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

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 specified by `first-col` and `last-col`). With no surprise, that command is available only in the environments without delimiters, that is to say `{NiceTabular}`, `{NiceArray}` and `{NiceMatrix}`.

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

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

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

^a Achard is an old family of the Poitou.

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

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

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

- **New 6.8** Since the version 6.8, the duplicates in the notes of a tabular are detected by default: if several commands `\tabularnote` are used in a tabular with the same argument, only one note is inserted at the end of the tabular (but all the labels are composed, of course). It's possible to de-activate that feature with the key `notes/detect-duplicates` (whose initial value is `true`).

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`.

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

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

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

13.2 Alignment option in `{NiceMatrix}`

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

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

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

13.3 The command `\rotate`

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

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

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

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

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

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

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

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

image of e_1 e_2 e_3

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`).

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

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

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

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

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

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.

```

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

```

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

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

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

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

```

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

```

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

13.6 The option `light-syntax`

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

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

```

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

```

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

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 \\
3 & 4
\end{bNiceMatrix}

```

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

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

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.

13.8 The environment `{NiceArrayWithDelims}`

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

```

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

```

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

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.

```

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

```

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

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

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).

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

```

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

```

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

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

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

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 interessant 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.

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

The names of the “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

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

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

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

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.

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

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

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

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.

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

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

```
\begin{NiceArray}{c@{\;}c@{\;}c@{\;}c}[create-medium-nodes]
  u_1 & - & u_0 & = & r & \\
  u_2 & - & u_1 & = & r & \\
  u_3 & - & u_2 & = & r & \\
  u_4 & - & u_3 & = & r & \\
  \phantom{u_5} & & \phantom{u_4} & & \smash{\vdots} & \\
  u_n & - & u_{n-1} & = & r & \\
\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`.

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

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).

```

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

```

$$\begin{pmatrix} 2 & 1 & 3 & | & 0 \\ 3 & 3 & 1 & | & 0 \\ \hline 3 & 3 & 1 & | & 0 \end{pmatrix}$$

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}\}`).

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

15 API for the developpers

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).

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

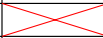
It’s possible to program such command `\crossbox` as follows, explicitly using the public variable `\g_nicematrix_code_after_tl`.

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

\NewDocumentCommand \crossbox { ! 0 { } }
{
  \tl_gput_right:Nx \g_nicematrix_code_after_tl
  {
    \__pantigny_crossbox:nnn
    { \int_use:c { c@iRow } }
    { \int_use:c { c@jCol } }
    { \exp_not:n { #1 } }
  }
}
\ExplSyntaxOff
```

Here is an example of utilisation:

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

merlan	requin	cabillaud
baleine		morue
mante	raie	poule

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.

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):

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

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

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

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

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

The same example without parallelization:

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

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.

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

- 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}` 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}` (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 `ieeaccess` (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.

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).*

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>

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}
& \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 argument⁵⁵

```
\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}`.

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

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

⁵⁵In fact: the value of its argument.

```

\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 \\
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 & & & & b_0 & & & \\
a_1 & & \Ddots & & b_1 & & \Ddots & \\
\vdots & & \Ddots & & \vdots & & \Ddots & b_0 \\
a_p & & & a_0 & & & b_1 & \\
& & \Ddots & & a_1 & & b_q & & \Ddots \\
& & & \Ddots & & & \Ddots & & \\
& & & a_p & & & & & b_q
\end{vNiceArray}

```


$$\left| \begin{array}{ccc} a_0 & & \\ & \ddots & \\ a_1 & & a_0 \\ & \ddots & \\ a_p & & a_1 \\ & \ddots & \\ & & a_p \end{array} \right| \begin{array}{ccc} b_0 & & \\ & \ddots & \\ b_1 & & b_0 \\ & \ddots & \\ b_q & & b_1 \\ & \ddots & \\ & & b_q \end{array} \right|$$

An example for a linear system:

```

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

```

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

17.4 Dotted lines which are no longer dotted

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

```

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

```

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

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

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

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}
```

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

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

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}
```

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

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

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

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

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

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

```

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

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

...
\end{NiceMatrixBlock}

```

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

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

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

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

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

```

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

```

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

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

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

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

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

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

```

\setlength{\extrarowheight}{1mm}
\[\begin{NiceArray}
[
vlines-in-sub-matrix=I,
last-col,
code-for-last-col = \scriptstyle \color{blue}
]
{rrrrIr}
12 & -8 & 7 & 5 & 3 & \\
3 & -18 & 12 & 1 & 4 & \\

```

```

-3 & -46 & 29 & -2 & -15 \\
 9 & 10 & & -5 & 4 & 7 \\[1mm]
12 & -8 & & 7 & 5 & 3 \\
0 & 64 & & -41 & 1 & 19 & L_2 \gets L_1-4L_2 \\
0 & & -192 & 123 & & -3 & -57 & L_3 \gets L_1+4L_3 \\
0 & & & -64 & 41 & & -1 & -19 & L_4 \gets 3L_1-4L_4 \\
12 & -8 & & 7 & 5 & 3 \\
0 & 64 & & -41 & 1 & 19 \\
0 & & 0 & & 0 & & 0 & & L_3 \gets 3L_2+L_3 \\
12 & -8 & & 7 & 5 & 3 \\
0 & 64 & & -41 & 1 & 19 \\
\CodeAfter
\SubMatrix({1-1}{4-5})
\SubMatrix({5-1}{8-5})
\SubMatrix({9-1}{11-5})
\SubMatrix({12-1}{13-5})
\end{NiceArray}\]

```

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

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

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

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

17.7 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).

```

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

```

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

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

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

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

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

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

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

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}}
```

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

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

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

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

```

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

```

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

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`.

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


```

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

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

```

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_generate_variant:Nn \@@_error:nn { n x }
15 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
16 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
17 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
18 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
19 \cs_new_protected:Npn \@@_msg_new:nnn { \msg_new:nnnn { nicematrix } }

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

```

Technical definitions

```

22 \tl_new:N \l_@@_argspec_tl

23 \cs_generate_variant:Nn \seq_gset_split:Nnn { N V n }
24 \cs_generate_variant:Nn \keys_define:nn { n x }

25 \hook_gput_code:nnn { begindocument } { . }
26 {
27   \@ifpackageloaded { varwidth }
28     { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_true_bool } }
29     { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_false_bool } }
30   \@ifpackageloaded { arydshln }
31     { \bool_const:Nn \c_@@_arydshln_loaded_bool { \c_true_bool } }
32     { \bool_const:Nn \c_@@_arydshln_loaded_bool { \c_false_bool } }
33   \@ifpackageloaded { booktabs }
34     { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_true_bool } }
35     { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_false_bool } }
36   \@ifpackageloaded { enumitem }
37     { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_true_bool } }
38     { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_false_bool } }
39   \@ifpackageloaded { tabularx }
40     { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_true_bool } }
41     { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_false_bool } }
42   { }
43   \@ifpackageloaded { tikz }
44   {

```

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.

```

45     \bool_const:Nn \c_@@_tikz_loaded_bool \c_true_bool
46     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
47     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
48   }
49   {
50     \bool_const:Nn \c_@@_tikz_loaded_bool \c_false_bool
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 2022, the current version `revtex4-2` is 4.2e (compatible with `booktabs`).

```

55 \ifclassloaded { revtex4-1 }
56 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
57 {
58   \ifclassloaded { revtex4-2 }
59   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
60   {

```

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

```

61     \cs_if_exist:NT \rvtx@ifformat@geq
62     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
63     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
64   }
65 }

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

```

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

```

67 \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }

```

If the final user uses `nicematrix`, PGF/Tikz will write instruction `\pgfsyspdfmark` in the aux file. If he changes its mind and no longer loads `nicematrix`, an error may occur at the next compilation because of remanent instructions `\pgfsyspdfmark` in the aux file. With the following code, we try to avoid that situation.

```

68 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
69 {
70   \iow_now:Nn \@mainaux
71   {
72     \ExplSyntaxOn
73     \cs_if_free:NT \pgfsyspdfmark
74     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
75     \ExplSyntaxOff
76   }
77   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
78 }

```

We define a command `\iddots` similar to `\ddots` (`\ddots`) but with dots going forward (`\iddots`). We use `\ProvideDocumentCommand` and so, if the command `\iddots` has already been defined (for example by the package `mathdots`), we don’t define it again.

```

79 \ProvideDocumentCommand \iddots { }
80 {
81   \mathinner
82   {

```

```

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

```

This definition is a variant of the standard definition of `\ddots`.

In the `aux` file, we will have the references of the PGF/Tikz nodes created by `nicematrix`. However, when `booktabs` is used, some nodes (more precisely, some `row` nodes) will be defined twice because their position will be modified. In order to avoid an error message in this case, we will redefine `\pgfutil@check@rerun` in the `aux` file.

```

93 \hook_gput_code:nnn { begindocument } { . }
94 {
95   \@ifpackageloaded { booktabs }
96   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
97   { }
98 }
99 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
100 {
101   \cs_set_eq:NN \@_old_pgful@check@rerun \pgfutil@check@rerun

```

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

```

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

```

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

```

108 \bool_new:N \l_@@_colortbl_loaded_bool
109 \hook_gput_code:nnn { begindocument } { . }
110 {
111   \@ifpackageloaded { colortbl }
112   { \bool_set_true:N \l_@@_colortbl_loaded_bool }
113   {

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. We will use it to store the instruction of color for the rules even if `colortbl` is not loaded.

```

114   \cs_set_protected:Npn \CT@arc@ { }
115   \cs_set:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
116   \cs_set:Npn \CT@arc #1 #2
117   {
118     \dim_compare:nNt \baselineskip = \c_zero_dim \noalign
119     { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
120   }

```

Idem for `\CT@drs@`.

```

121   \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
122   \cs_set:Npn \CT@drs #1 #2
123   {
124     \dim_compare:nNt \baselineskip = \c_zero_dim \noalign
125     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
126   }
127   \cs_set:Npn \hline
128   {
129     \noalign { \ifnum 0 = ` } \fi
130     \cs_set_eq:NN \hskip \vskip

```

```

131         \cs_set_eq:NN \vrule \hrule
132         \cs_set_eq:NN \@width \@height
133         { \CT@arc@ \vline }
134         \futurelet \reserved@a
135         \@xhline
136     }
137 }
138 }

```

We 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.

```

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

```

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

```

148     \skip_horizontal:N \c_zero_dim
149 }

```

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`.

```

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

```

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.

```

154 \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`.

```

155 { \@@_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*.

```

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

```

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).

```

167     \int_compare:nNnT { #1 } < { #2 }
168     { \multispan { \int_eval:n { #2 - #1 } } & }
169     \multispan { \int_eval:n { #3 - #2 + 1 } }

```

⁵⁹See question 99041 on TeX StackExchange.

```

170 {
171   \CT@arc@
172   \leaders \hrule \@height \arrayrulewidth \hfill
173   \skip_horizontal:N \c_zero_dim
174 }

```

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

```

175 \peek_meaning_remove_ignore_spaces:NTF \cline
176 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
177 { \everycr { } \cr }
178 }
179 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

The following command is a small shortcut.

```

180 \cs_new:Npn \@@_math_toggle_token:
181 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

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

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

194 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The column S of siunitx

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

```

195 \bool_new:N \l_@@_siunitx_loaded_bool
196 \hook_gput_code:nnn { begindocument } { . }
197 {
198   \@ifpackageloaded { siunitx }
199   { \bool_set_true:N \l_@@_siunitx_loaded_bool }
200   { }
201 }

```

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

```

202 \hook_gput_code:nnn { begindocument } { . }
203 {
204   \bool_if:NTF { ! \l_@@_siunitx_loaded_bool }
205   { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
206   {
207     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
208     {
209       \renewcommand*{\NC@rewrite@S}[1] []
210       {

```

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

```

211       \@temptokena \exp_after:wN
212       { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
213       \NC@find
214     }
215   }

```

```

216     }
217 }

```

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.

```

218 \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).

```

219 \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.

```

220 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
221 { \int_use:N \g_@@_env_int }

```

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

```

222 \cs_new_protected:Npn \@@_qpoint:n #1
223 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

224 \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).

```

225 \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.).

```

226 \dim_new:N \l_@@_col_width_dim
227 \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.

```

228 \int_new:N \g_@@_row_total_int
229 \int_new:N \g_@@_col_total_int

```

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

```

230 \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.

```

231 \str_new:N \l_@@_hpos_cell_str
232 \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`.

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

Idem pour the mono-row blocks.

```
234 \dim_new:N \g_@@_blocks_ht_dim
```

```
235 \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}`).

```
236 \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`.

```
237 \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.

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

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

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

```
240 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

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

```
241 \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.

```
242 \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.

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

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

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

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

```
245 \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.

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

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

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



```

248 \cs_new_protected:Npn \@@_test_if_math_mode:
249 {
250   \if_mode_math: \else:
251     \@@_fatal:n { Outside-math-mode }
252   \fi:
253 }

```

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

```

254 \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.

```

255 \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”.

```

256 \colorlet { nicematrix-last-col } { . }
257 \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*).

```

258 \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*.

```

259 \tl_new:N \g_@@_com_or_env_str
260 \tl_gset:Nn \g_@@_com_or_env_str { environment }

```

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

```

261 \cs_new:Npn \@@_full_name_env:
262 {
263   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
264     { command \space \c_backslash_str \g_@@_name_env_str }
265     { environment \space \{ \g_@@_name_env_str \} }
266 }

```

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*.

```

267 \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.

```

268 \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.

```

269 \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.

```
270 \int_new:N \l_@@_old_iRow_int
271 \int_new:N \l_@@_old_jCol_int
```

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

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

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

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

```
273 \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.

```
274 \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.

```
275 \bool_new:N \l_@@_X_columns_aux_bool
276 \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.

```
277 \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).

```
278 \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).

```
279 \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`).

```
280 \tl_new:N \l_@@_code_before_tl
281 \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).

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

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

```
283 \dim_new:N \l_@@_x_initial_dim
284 \dim_new:N \l_@@_y_initial_dim
285 \dim_new:N \l_@@_x_final_dim
286 \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.

```
287 \dim_zero_new:N \l_@@_tmpc_dim
288 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
289 \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”.

```
290 \dim_new:N \g_@@_width_last_col_dim
291 \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.

```
292 \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!).

```
293 \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}`.

```
294 \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.

```
295 \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.

```
296 \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`).

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

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

```
298 \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).

```
299 \seq_new:N \g_@@_multicolumn_cells_seq
300 \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`).

```
301 \int_new:N \l_@@_row_min_int
302 \int_new:N \l_@@_row_max_int
303 \int_new:N \l_@@_col_min_int
304 \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.

```
305 \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).

```
306 \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`.

```
307 \tl_new:N \l_@@_fill_tl
308 \tl_new:N \l_@@_draw_tl
309 \seq_new:N \l_@@_tikz_seq
310 \clist_new:N \l_@@_borders_clist
311 \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`.

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

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

```
313 \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`).

```
314 \str_new:N \l_@@_hpos_block_str
315 \str_set:Nn \l_@@_hpos_block_str { c }
316 \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`.

```
317 \tl_new:N \l_@@_vpos_of_block_tl
318 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

```
319 \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).

```
320 \bool_new:N \l_@@_vlines_block_bool
321 \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.

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

323 \dim_new:N \l_@@_submatrix_extra_height_dim
324 \dim_new:N \l_@@_submatrix_left_xshift_dim
325 \dim_new:N \l_@@_submatrix_right_xshift_dim
326 \clist_new:N \l_@@_hlines_clist
327 \clist_new:N \l_@@_vlines_clist
328 \clist_new:N \l_@@_submatrix_hlines_clist
329 \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.

```
330 \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.

```
331 \int_new:N \l_@@_first_row_int
332 \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.

```
333 \int_new:N \l_@@_first_col_int
334 \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).

```
335 \int_new:N \l_@@_last_row_int
336 \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

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

⁶⁰We can’t use `\l_@@_last_row_int` for this usage because, if `nicematrix` has read its value from the `aux` file, the value of the counter won’t be `-1` any longer.

Idem for `\l_@@_last_col_without_value_bool`

```
338 \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.

```
339 \int_new:N \l_@@_last_col_int
340 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

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

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

Some utilities

```
342 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
343 {
344   \tl_set:Nn \l_tmpa_tl { #1 }
345   \tl_set:Nn \l_tmpb_tl { #2 }
346 }
```

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.

```
347 \cs_new_protected:Npn \@@_expand_clist:N #1
348 {
349   \clist_if_in:NnF #1 { all }
350   {
351     \clist_clear:N \l_tmpa_clist
352     \clist_map_inline:Nn #1
353     {
354       \tl_if_in:nnTF { ##1 } { - }
355       { \@@_cut_on_hyphen:w ##1 \q_stop }
356       {
357         \tl_set:Nn \l_tmpa_tl { ##1 }
358         \tl_set:Nn \l_tmpb_tl { ##1 }
359       }
360       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
361       { \clist_put_right:Nn \l_tmpa_clist { ###1 } }
362     }
363     \tl_set_eq:NN #1 \l_tmpa_clist
364   }
365 }
```

The command `\tabularnote`

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

```
366 \newcounter { tabularnote }
```

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

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

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

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

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

```
370 \newcounter{nicematrix_draft}  
371 \cs_new_protected:Npn \@@_notes_format:n #1  
372 {  
373   \setcounter { nicematrix_draft } { #1 }  
374   \@@_notes_style:n { nicematrix_draft }  
375 }
```

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

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

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

```
377 \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`.

```
378 \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`.

```
379 \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`).

```
380 \hook_gput_code:nnn { begindocument } { . }  
381 {  
382   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }  
383   {  
384     \NewDocumentCommand \tabularnote { m }  
385     { \@@_error:n { enumitem-not-loaded } }  
386   }  
387 }
```

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.

```

388     \newlist { tabularnotes } { enumerate } { 1 }
389     \setlist [ tabularnotes ]
390     {
391         topsep = Opt ,
392         noitemsep ,
393         leftmargin = * ,
394         align = left ,
395         labelsep = Opt ,
396         label =
397             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
398     }
399     \newlist { tabularnotes* } { enumerate* } { 1 }
400     \setlist [ tabularnotes* ]
401     {
402         afterlabel = \nobreak ,
403         itemjoin = \quad ,
404         label =
405             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
406     }

```

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

```

407     \NewDocumentCommand \tabularnote { m }
408     {
409         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
410         { \@@_error:n { tabularnote~forbidden } }
411         {

```

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

```

412         \int_zero:N \l_tmpa_int
413         \bool_if:NT \l_@@_notes_detect_duplicates_bool
414         {
415             \seq_map_indexed_inline:Nn \g_@@_tabularnotes_seq
416             {
417                 \tl_if_eq:nnT { #1 } { ##2 }
418                 { \int_set:Nn \l_tmpa_int { ##1 } \seq_map_break: }
419             }
420         }
421         \int_compare:nNnTF \l_tmpa_int = 0
422         {
423             \stepcounter { tabularnote }
424             \seq_put_right:Nx \l_@@_notes_labels_seq
425             { \@@_notes_format:n { \int_use:c { c @ tabularnote } } }
426             \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
427         }
428         {
429             \seq_put_right:Nx \l_@@_notes_labels_seq

```

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


```

430         { \@@_notes_format:n { \int_use:N \l_tmpa_int } }
431     }
432     \peek_meaning:NF \tabularnote
433     {

```

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.

```

434         \hbox_set:Nn \l_tmpa_box
435     {

```

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

```

436         \@@_notes_label_in_tabular:n
437         {
438             \seq_use:Nnnn
439             \l_@@_notes_labels_seq { , } { , } { , }
440         }
441     }

```

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.

```

442         \addtocounter { tabularnote } { -1 }
443         \refstepcounter { tabularnote }
444         \seq_clear:N \l_@@_notes_labels_seq
445         \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).

```

446         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
447     }
448 }
449 }
450 }
451 }

```

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.

```

452 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
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 { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
462     \pgfnode
463     { rectangle }
464     { center }
465     {
466         \vbox_to_ht:nn
467         { \dim_abs:n { #5 - #3 } }
468         {
469             \vfill

```

```

470         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
471     }
472 }
473 { #1 }
474 { }
475 \end { pgfscope }
476 }

```

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.

```

477 \cs_new_protected:Npn \@@pgf_rect_node:nnn #1 #2 #3
478 {
479     \begin { pgfscope }
480     \pgfset
481     {
482         outer~sep = \c_zero_dim ,
483         inner~sep = \c_zero_dim ,
484         minimum~size = \c_zero_dim
485     }
486     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
487     \pgfpointdiff { #3 } { #2 }
488     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
489     \pgfnode
490     { rectangle }
491     { center }
492     {
493         \vbox_to_ht:nn
494         { \dim_abs:n \l_tmpb_dim }
495         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
496     }
497     { #1 }
498     { }
499     \end { pgfscope }
500 }

```

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.

```

501 \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`.

```

502 \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`).

```

503 \dim_new:N \l_@@_cell_space_top_limit_dim
504 \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.

```

505 \dim_new:N \l_@@_inter_dots_dim
506 \hook_gput_code:nnn { begindocument } { . }
507 { \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).

```
508 \dim_new:N \l_@@_xdots_shorten_dim
509 \hook_gput_code:nnn { begindocument } { . }
510 { \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.

```
511 \dim_new:N \l_@@_radius_dim
512 \hook_gput_code:nnn { begindocument } { . }
513 { \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.

```
514 \tl_new:N \l_@@_xdots_line_style_tl
515 \tl_const:Nn \c_@@_standard_tl { standard }
516 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
517 \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).

```
518 \tl_new:N \l_@@_baseline_tl
519 \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`).

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

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

```
521 \bool_new:N \l_@@_parallelize_diags_bool
522 \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.

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

```
524 \dim_new:N \l_@@_notes_above_space_dim
525 \hook_gput_code:nnn { begindocument } { . }
526 { \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.).

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

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

```
528 \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).

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

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

```
530 \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.

```
531 \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”.

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

```
533 \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).

```
534 \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`).

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

```
536 \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`.

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

```
538 \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.

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

```
540 \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 “:”.

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

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

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

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

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

```
544 \keys_define:nn { NiceMatrix / xdots }
545 {
546   line-style .code:n =
547   {
548     \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.

```
549     { \cs_if_exist_p:N \tikzpicture }
550     { \str_if_eq_p:nn { #1 } { standard } }
551     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
552     { \@@_error:n { bad-option-for~line-style } }
553   } ,
554   line-style .value_required:n = true ,
555   color .tl_set:N = \l_@@_xdots_color_tl ,
556   color .value_required:n = true ,
557   shorten .code:n =
558     \hook_gput_code:nnn { begindocument } { . }
559     { \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).

```
560   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 `_`.

```
561   down .tl_set:N = \l_@@_xdots_down_tl ,
562   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).

```
563   draw-first .code:n = \prg_do_nothing: ,
564   unknown .code:n = \@@_error:n { Unknown-key-for~xdots }
565 }
```

```
566 \keys_define:nn { NiceMatrix / rules }
567 {
568   color .tl_set:N = \l_@@_rules_color_tl ,
569   color .value_required:n = true ,
570   width .dim_set:N = \arrayrulewidth ,
571   width .value_required:n = true
572 }
```

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

```
573 \keys_define:nn { NiceMatrix / Global }
574 {
575   custom-line .code:n = \@@_custom_line:n { #1 } ,
576   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
577   delimiters .value_required:n = true ,
578   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
579   rules .value_required:n = true ,
```

```

580 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
581 standard-cline .default:n = true ,
582 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
583 cell-space-top-limit .value_required:n = true ,
584 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
585 cell-space-bottom-limit .value_required:n = true ,
586 cell-space-limits .meta:n =
587 {
588     cell-space-top-limit = #1 ,
589     cell-space-bottom-limit = #1 ,
590 } ,
591 cell-space-limits .value_required:n = true ,
592 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
593 light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
594 light-syntax .default:n = true ,
595 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
596 end-of-row .value_required:n = true ,
597 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
598 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
599 last-row .int_set:N = \l_@@_last_row_int ,
600 last-row .default:n = -1 ,
601 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
602 code-for-first-col .value_required:n = true ,
603 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
604 code-for-last-col .value_required:n = true ,
605 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
606 code-for-first-row .value_required:n = true ,
607 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
608 code-for-last-row .value_required:n = true ,
609 hlines .clist_set:N = \l_@@_hlines_clist ,
610 vlines .clist_set:N = \l_@@_vlines_clist ,
611 hlines .default:n = all ,
612 vlines .default:n = all ,
613 vlines-in-sub-matrix .code:n =
614 {
615     \tl_if_single_token:nTF { #1 }
616     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
617     { \@@_error:n { One~letter~allowed } }
618 } ,
619 vlines-in-sub-matrix .value_required:n = true ,
620 hvlines .code:n =
621 {
622     \clist_set:Nn \l_@@_vlines_clist { all }
623     \clist_set:Nn \l_@@_hlines_clist { all }
624 } ,
625 hvlines-except-borders .code:n =
626 {
627     \clist_set:Nn \l_@@_vlines_clist { all }
628     \clist_set:Nn \l_@@_hlines_clist { all }
629     \bool_set_true:N \l_@@_except_borders_bool
630 } ,
631 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.

```

632 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
633 renew-dots .value_forbidden:n = true ,
634 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
635 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
636 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
637 create-extra-nodes .meta:n =
638 { create-medium-nodes , create-large-nodes } ,
639 left-margin .dim_set:N = \l_@@_left_margin_dim ,

```

```

640 left-margin .default:n = \arraycolsep ,
641 right-margin .dim_set:N = \l_@@_right_margin_dim ,
642 right-margin .default:n = \arraycolsep ,
643 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
644 margin .default:n = \arraycolsep ,
645 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
646 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
647 extra-margin .meta:n =
648   { extra-left-margin = #1 , extra-right-margin = #1 } ,
649 extra-margin .value_required:n = true ,
650 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
651 respect-arraystretch .default:n = true
652 }

```

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

```

653 \keys_define:nn { NiceMatrix / Env }
654 {

```

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

```

655   hvlines-except-corners .code:n =
656   {
657     \@@_error:n { hvlines-except-corners }
658     \group_begin:
659     \globaldefs = 1
660     \@@_msg_redirect_name:nn { hvlines-except-corners } { none }
661     \group_end:
662     \clist_set:Nn \l_@@_corners_clist { #1 }
663     \clist_set:Nn \l_@@_vlines_clist { all }
664     \clist_set:Nn \l_@@_hlines_clist { all }
665   } ,
666   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
667   corners .clist_set:N = \l_@@_corners_clist ,
668   corners .default:n = { NW , SW , NE , SE } ,
669   code-before .code:n =
670   {
671     \tl_if_empty:nF { #1 }
672     {
673       \tl_put_right:Nn \l_@@_code_before_tl { #1 }
674       \bool_set_true:N \l_@@_code_before_bool
675     }
676   } ,

```

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

```

677   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
678   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
679   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
680   baseline .tl_set:N = \l_@@_baseline_tl ,
681   baseline .value_required:n = true ,
682   columns-width .code:n =
683   { \tl_if_eq:nnTF { #1 } { auto }
684     { \bool_set_true:N \l_@@_auto_columns_width_bool }
685     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } } ,
686   columns-width .value_required:n = true ,
687   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.

```

688   \legacy_if:nF { measuring@ }
689   {
690     \str_set:Nn \l_tmpa_str { #1 }
691     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str

```

```

692         { \@@_error:nn { Duplicate-name } { #1 } }
693         { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
694         \str_set_eq:NN \l_@@_name_str \l_tmpa_str
695     } ,
696     name .value_required:n = true ,
697     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
698     code-after .value_required:n = true ,
699     colortbl-like .code:n =
700         \bool_set_true:N \l_@@_colortbl_like_bool
701         \bool_set_true:N \l_@@_code_before_bool ,
702     colortbl-like .value_forbidden:n = true
703 }
704 \keys_define:nn { NiceMatrix / notes }
705 {
706     para .bool_set:N = \l_@@_notes_para_bool ,
707     para .default:n = true ,
708     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
709     code-before .value_required:n = true ,
710     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
711     code-after .value_required:n = true ,
712     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
713     bottomrule .default:n = true ,
714     style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
715     style .value_required:n = true ,
716     label-in-tabular .code:n =
717         \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
718     label-in-tabular .value_required:n = true ,
719     label-in-list .code:n =
720         \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
721     label-in-list .value_required:n = true ,
722     enumitem-keys .code:n =
723     {
724         \hook_gput_code:nnn { begindocument } { . }
725         {
726             \bool_if:NT \c_@@_enumitem_loaded_bool
727             { \setlist* [ tabularnotes ] { #1 } }
728         }
729     } ,
730     enumitem-keys .value_required:n = true ,
731     enumitem-keys-para .code:n =
732     {
733         \hook_gput_code:nnn { begindocument } { . }
734         {
735             \bool_if:NT \c_@@_enumitem_loaded_bool
736             { \setlist* [ tabularnotes* ] { #1 } }
737         }
738     } ,
739     enumitem-keys-para .value_required:n = true ,
740     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
741     detect-duplicates .default:n = true ,
742     unknown .code:n = \@@_error:n { Unknown-key-for-notes }
743 }
744 \keys_define:nn { NiceMatrix / delimiters }
745 {
746     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
747     max-width .default:n = true ,
748     color .tl_set:N = \l_@@_delimiters_color_tl ,
749     color .value_required:n = true ,
750 }

```

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


```

751 \keys_define:nn { NiceMatrix }
752 {
753   NiceMatrixOptions .inherit:n =
754     { NiceMatrix / Global } ,
755   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
756   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
757   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
758   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
759   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
760   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
761   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
762   NiceMatrix .inherit:n =
763     {
764       NiceMatrix / Global ,
765       NiceMatrix / Env ,
766     } ,
767   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
768   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
769   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
770   NiceTabular .inherit:n =
771     {
772       NiceMatrix / Global ,
773       NiceMatrix / Env
774     } ,
775   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
776   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
777   NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
778   NiceArray .inherit:n =
779     {
780       NiceMatrix / Global ,
781       NiceMatrix / Env ,
782     } ,
783   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
784   NiceArray / rules .inherit:n = NiceMatrix / rules ,
785   NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
786   pNiceArray .inherit:n =
787     {
788       NiceMatrix / Global ,
789       NiceMatrix / Env ,
790     } ,
791   pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
792   pNiceArray / rules .inherit:n = NiceMatrix / rules ,
793   pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
794 }

```

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

```

795 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
796 {
797   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
798   width .value_required:n = true ,
799   last-col .code:n = \tl_if_empty:nF { #1 }
800     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
801     \int_zero:N \l_@@_last_col_int ,
802   small .bool_set:N = \l_@@_small_bool ,
803   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.

```

804   renew-matrix .code:n = \@@_renew_matrix: ,
805   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}`.

```
806   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.

```
807   columns-width .code:n =
808     \tl_if_eq:nnTF { #1 } { auto }
809     { \@@_error:n { Option~auto~for~columns-width } }
810     { \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.

```
811   allow-duplicate-names .code:n =
812     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
813   allow-duplicate-names .value_forbidden:n = true ,
```

The key `letter-for-dotted-lines` is now obsolete. You will delete it in a future version.

```
814   letter-for-dotted-lines .code:n =
815   {
816     \@@_error:n { letter-for-dotted-lines }
817     \group_begin:
818     \globaldefs = 1
819     \@@_msg_redirect_name:nn { letter-for-dotted-lines } { none }
820     \group_end:
821     \tl_if_single_token:nTF { #1 }
822     { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
823     { \@@_error:n { One-letter~allowed } }
824   } ,
825   letter-for-dotted-lines .value_required:n = true ,
826   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
827   notes .value_required:n = true ,
828   sub-matrix .code:n =
829     \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
830   sub-matrix .value_required:n = true ,
831   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
832 } ,
```

The following string will initially be empty. It will be set by the key `'letter-for-dotted-lines'`.

```
833 \str_new:N \l_@@_letter_for_dotted_lines_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.

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

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

```
836 \keys_define:nn { NiceMatrix / NiceMatrix }
837 {
838   last-col .code:n = \tl_if_empty:nTF {#1}
839   {
840     \bool_set_true:N \l_@@_last_col_without_value_bool
841     \int_set:Nn \l_@@_last_col_int { -1 }
842   }
843   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
844   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
845   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
846   small .bool_set:N = \l_@@_small_bool ,
```

```

847     small .value_forbidden:n = true ,
848     unknown .code:n = \@@_error:n { Unknown-key-for~NiceMatrix }
849 }

```

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

```

850 \keys_define:nn { NiceMatrix / NiceArray }
851 {

```

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.

```

852     small .bool_set:N = \l_@@_small_bool ,
853     small .value_forbidden:n = true ,
854     last-col .code:n = \tl_if_empty:nF { #1 }
855         { \@@_error:n { last-col~non-empty~for~NiceArray } }
856         \int_zero:N \l_@@_last_col_int ,
857     notes / para .bool_set:N = \l_@@_notes_para_bool ,
858     notes / para .default:n = true ,
859     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
860     notes / bottomrule .default:n = true ,
861     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
862     tabularnote .value_required:n = true ,
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~NiceArray }
866 }
867 \keys_define:nn { NiceMatrix / pNiceArray }
868 {
869     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
870     last-col .code:n = \tl_if_empty:nF {#1}
871         { \@@_error:n { last-col~non-empty~for~NiceArray } }
872         \int_zero:N \l_@@_last_col_int ,
873     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
874     small .bool_set:N = \l_@@_small_bool ,
875     small .value_forbidden:n = true ,
876     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
877     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
878     unknown .code:n = \@@_error:n { Unknown-key-for~NiceMatrix }
879 }

```

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

```

880 \keys_define:nn { NiceMatrix / NiceTabular }
881 {

```

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.

```

882     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
883         \bool_set_true:N \l_@@_width_used_bool ,
884     width .value_required:n = true ,
885     notes / para .bool_set:N = \l_@@_notes_para_bool ,
886     notes / para .default:n = true ,
887     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
888     notes / bottomrule .default:n = true ,
889     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
890     tabularnote .value_required:n = true ,
891     last-col .code:n = \tl_if_empty:nF {#1}
892         { \@@_error:n { last-col~non-empty~for~NiceArray } }
893         \int_zero:N \l_@@_last_col_int ,
894     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
895     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
896     unknown .code:n = \@@_error:n { Unknown-key-for~NiceTabular }
897 }

```

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}`).

```
898 \cs_new_protected:Npn \@@_cell_begin:w
899 {
```

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*).

```
900 \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).

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

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

```
902 \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.

```
903 \int_compare:nNnT \c@jCol = 1
904 { \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).

```
905 \hbox_set:Nw \l_@@_cell_box
906 \bool_if:NF \l_@@_NiceTabular_bool
907 {
908   \c_math_toggle_token
909   \bool_if:NT \l_@@_small_bool \scriptstyle
910 }
```

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).

```
911 \color { nicematrix }
```

```
912 \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.

```
913 \int_compare:nNnTF \c@iRow = 0
914 {
915   \int_compare:nNnT \c@jCol > 0
916   {
917     \l_@@_code_for_first_row_tl
918     \xglobal \colorlet { nicematrix-first-row } { . }
919   }
920 }
921 {
922   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
923   {
924     \l_@@_code_for_last_row_tl
925     \xglobal \colorlet { nicematrix-last-row } { . }
926   }
927 }
928 }
```

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.

```

929 \cs_new_protected:Npn \@@_begin_of_row:
930 {
931   \int_gincr:N \c@iRow
932   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
933   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
934   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
935   \pgfpicture
936   \pgfrememberpicturepositiononpagetrue
937   \pgfcoordinate
938   { \@@_env: - row - \int_use:N \c@iRow - base }
939   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
940   \str_if_empty:NF \l_@@_name_str
941   {
942     \pgfnodealias
943     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
944     { \@@_env: - row - \int_use:N \c@iRow - base }
945   }
946   \endpgfpicture
947 }

```

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.

```

948 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
949 {
950   \int_compare:nNnTF \c@iRow = 0
951   {
952     \dim_gset:Nn \g_@@_dp_row_zero_dim
953     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
954     \dim_gset:Nn \g_@@_ht_row_zero_dim
955     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
956   }
957   {
958     \int_compare:nNnT \c@iRow = 1
959     {
960       \dim_gset:Nn \g_@@_ht_row_one_dim
961       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
962     }
963   }
964 }

965 \cs_new_protected:Npn \@@_rotate_cell_box:
966 {
967   \box_rotate:Nn \l_@@_cell_box { 90 }
968   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
969   {
970     \vbox_set_top:Nn \l_@@_cell_box
971     {
972       \vbox_to_zero:n { }
973       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
974       \box_use:N \l_@@_cell_box
975     }
976   }
977   \bool_gset_false:N \g_@@_rotate_bool
978 }

979 \cs_new_protected:Npn \@@_adjust_size_box:
980 {
981   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim

```

```

982     {
983       \box_set_wd:Nn \l_@@_cell_box
984       { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
985       \dim_gzero:N \g_@@_blocks_wd_dim
986     }
987     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
988     {
989       \box_set_dp:Nn \l_@@_cell_box
990       { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
991       \dim_gzero:N \g_@@_blocks_dp_dim
992     }
993     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
994     {
995       \box_set_ht:Nn \l_@@_cell_box
996       { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
997       \dim_gzero:N \g_@@_blocks_ht_dim
998     }
999   }
1000   \cs_new_protected:Npn \@@_cell_end:
1001   {
1002     \@@_math_toggle_token:
1003     \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.

```

1004   \g_@@_post_action_cell_tl
1005   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1006   \@@_adjust_size_box:
1007   \box_set_ht:Nn \l_@@_cell_box
1008   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1009   \box_set_dp:Nn \l_@@_cell_box
1010   { \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”).

```

1011   \dim_gset:Nn \g_@@_max_cell_width_dim
1012   { \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”.

```

1013   \@@_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.

```

1014 \bool_if:NTF \g_@@_empty_cell_bool
1015 { \box_use_drop:N \l_@@_cell_box }
1016 {
1017   \bool_lazy_or:nnTF
1018     \g_@@_not_empty_cell_bool
1019     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1020     \@@_node_for_cell:
1021     { \box_use_drop:N \l_@@_cell_box }
1022 }
1023 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1024 \bool_gset_false:N \g_@@_empty_cell_bool
1025 \bool_gset_false:N \g_@@_not_empty_cell_bool
1026 }

```

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

```

1027 \cs_new_protected:Npn \@@_node_for_cell:
1028 {
1029   \pgfpicture
1030   \pgfsetbaseline \c_zero_dim
1031   \pgfrememberpicturepositiononpagetrue
1032   \pgfset
1033   {
1034     inner~sep = \c_zero_dim ,
1035     minimum~width = \c_zero_dim
1036   }
1037   \pgfnode
1038   { rectangle }
1039   { base }
1040   { \box_use_drop:N \l_@@_cell_box }
1041   { \@@_env: - \int_use:N \c_iRow - \int_use:N \c_jCol }
1042   { }
1043   \str_if_empty:NF \l_@@_name_str
1044   {
1045     \pgfnodealias
1046     { \l_@@_name_str - \int_use:N \c_iRow - \int_use:N \c_jCol }
1047     { \@@_env: - \int_use:N \c_iRow - \int_use:N \c_jCol }
1048   }
1049   \endpgfpicture
1050 }

```

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.

```

1051 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1052 {
1053   \cs_new_protected:Npn \@@_patch_node_for_cell:
1054   {
1055     \hbox_set:Nn \l_@@_cell_box
1056     {
1057       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1058       \hbox_overlap_left:n
1059       {
1060         \pgfsys@markposition
1061         { \@@_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.

```

1062       #1
1063     }
1064     \box_use:N \l_@@_cell_box
1065     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1066     \hbox_overlap_left:n

```

```

1067         {
1068             \pgfsys@markposition
1069             { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1070             #1
1071         }
1072     }
1073 }
1074 }

```

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

```

1075 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1076 {
1077     \@@_patch_node_for_cell:n
1078     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1079 }
1080 { \@@_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.

```

1081 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1082 {
1083     \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1084     { \g_@@_#2 _ lines _ tl }
1085     {
1086         \use:c { @@ _ draw _ #2 : nnn }
1087         { \int_use:N \c@iRow }
1088         { \int_use:N \c@jCol }
1089         { \exp_not:n { #3 } }
1090     }
1091 }
1092 \cs_new_protected:Npn \@@_array:
1093 {
1094     \bool_if:NTF \l_@@_NiceTabular_bool
1095     { \dim_set_eq:NN \col@sep \tabcolsep }
1096     { \dim_set_eq:NN \col@sep \arraycolsep }
1097     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1098     { \cs_set_nopar:Npn \@halignto { } }
1099     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1100     \@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.

```

1101     [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1102 }

```


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.

```
1103 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1104 \cs_new_protected:Npn \@@_create_row_node:
1105 {
```

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

```
1106   \hbox
1107   {
1108     \bool_if:NT \l_@@_code_before_bool
1109     {
1110       \vtop
1111       {
1112         \skip_vertical:N 0.5\arrayrulewidth
1113         \pgfsys@markposition
1114         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1115         \skip_vertical:N -0.5\arrayrulewidth
1116       }
1117     }
1118     \pgfpicture
1119     \pgfrememberpicturepositiononpagetrue
1120     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1121     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1122     \str_if_empty:NF \l_@@_name_str
1123     {
1124       \pgfnodealias
1125       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1126       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1127     }
1128     \endpgfpicture
1129   }
1130 }
```

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

```
1131 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1132 \cs_new_protected:Npn \@@_everycr_i:
1133 {
1134   \int_gzero:N \c@jCol
1135   \bool_gset_false:N \g_@@_after_col_zero_bool
1136   \bool_if:NF \g_@@_row_of_col_done_bool
1137   {
1138     \@@_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).

```
1139     \tl_if_empty:NF \l_@@_hlines_clist
1140     {
1141       \tl_if_eq:NnF \l_@@_hlines_clist { all }
1142       {
1143         \exp_args:NNx
1144         \clist_if_in:NnT
1145         \l_@@_hlines_clist
1146         { \int_eval:n { \c@iRow + 1 } }
1147       }
1148     }
```

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.

```
1149     \int_compare:nNnT \c@iRow > { -1 }
1150     {
1151       \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@`.

```

1152         { \hrule height \arrayrulewidth width \c_zero_dim }
1153     }
1154 }
1155 }
1156 }
1157 }
```

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`).

```

1158 \cs_set_protected:Npn \@@_newcolumntype #1
1159 {
1160     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1161     \peek_meaning:NTF [
1162         { \newcol@ #1 }
1163         { \newcol@ #1 [ 0 ] }
1164     }
```

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

```

1165 \cs_set_protected:Npn \@@_renew_dots:
1166 {
1167     \cs_set_eq:NN \ldots \@@_Ldots
1168     \cs_set_eq:NN \cdots \@@_Cdots
1169     \cs_set_eq:NN \vdots \@@_Vdots
1170     \cs_set_eq:NN \ddots \@@_Ddots
1171     \cs_set_eq:NN \iddots \@@_Iddots
1172     \cs_set_eq:NN \dots \@@_Ldots
1173     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1174 }
```

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

```

1175 \cs_new_protected:Npn \@@_colortbl_like:
1176 {
1177     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1178     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1179     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1180 }
```

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

```

1181 \cs_new_protected:Npn \@@_pre_array_ii:
1182 {
```

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.

```

1183     \xglobal \colorlet { nicematrix } { . }
```

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

```

1184     \int_gzero:N \g_@@_total_X_weight_int
1185     \@@_expand_clist:N \l_@@_hlines_clist
1186     \@@_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.

```

1187 \bool_if:NT \c_@@_booktabs_loaded_bool
1188 { \tl_put_left:Nn \@BTnormal \@_create_row_node: }
1189 \box_clear_new:N \l_@@_cell_box
1190 \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}`).

```

1191 \bool_if:NT \l_@@_small_bool
1192 {
1193     \cs_set_nopar:Npn \arraystretch { 0.47 }
1194     \dim_set:Nn \arraycolsep { 1.45 pt }
1195 }

1196 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1197 {
1198     \tl_put_right:Nn \@_begin_of_row:
1199     {
1200         \pgfsys@markposition
1201         { \@_env: - row - \int_use:N \c@iRow - base }
1202     }
1203 }

```

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`.

```

1204 \cs_set_nopar:Npn \ialign
1205 {
1206     \bool_if:NTF \l_@@_colortbl_loaded_bool
1207     {
1208         \CT@everycr
1209         {
1210             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1211             \@_everycr:
1212         }
1213     }
1214     { \everycr { \@_everycr: } }
1215     \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`.

```

1216 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1217 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1218 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1219 \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}`.

```

1220 \dim_gzero_new:N \g_@@_ht_row_one_dim
1221 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1222 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1223 \dim_gzero_new:N \g_@@_ht_last_row_dim
1224 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1225 \dim_gzero_new:N \g_@@_dp_last_row_dim
1226 \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`.

```

1227 \cs_set_eq:NN \ialign \@@_old_ialign:
1228 \halign
1229 }

```

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).

```

1230 \cs_set_eq:NN \@@_old_ldots \ldots
1231 \cs_set_eq:NN \@@_old_cdots \cdots
1232 \cs_set_eq:NN \@@_old_vdots \vdots
1233 \cs_set_eq:NN \@@_old_ddots \ddots
1234 \cs_set_eq:NN \@@_old_iddots \iddots
1235 \bool_if:NTF \l_@@_standard_cline_bool
1236 { \cs_set_eq:NN \cline \@@_standard_cline }
1237 { \cs_set_eq:NN \cline \@@_cline }
1238 \cs_set_eq:NN \Ldots \@@_Ldots
1239 \cs_set_eq:NN \Cdots \@@_Cdots
1240 \cs_set_eq:NN \Vdots \@@_Vdots
1241 \cs_set_eq:NN \Ddots \@@_Ddots
1242 \cs_set_eq:NN \Iddots \@@_Iddots
1243 \cs_set_eq:NN \Hline \@@_Hline:
1244 \cs_set_eq:NN \Hspace \@@_Hspace:
1245 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1246 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1247 \cs_set_eq:NN \Block \@@_Block:
1248 \cs_set_eq:NN \rotate \@@_rotate:
1249 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1250 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1251 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1252 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1253 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1254 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1255 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1256 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1257 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1258 \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.

```

1259 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1260 \hook_gput_code:nnn { env / tabular / begin } { . }
1261 { \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).

```

1262 \seq_gclear:N \g_@@_multicolumn_cells_seq
1263 \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).

```

1264 \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`).

```
1265 \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.

```
1266 \int_gzero_new:N \g_@@_col_total_int
1267 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1268 \@@_renew_NC@rewrite@S:
1269 \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.

```
1270 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1271 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1272 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1273 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1274 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1275 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1276 \tl_gclear_new:N \g_nicematrix_code_before_tl
1277 }
```

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

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

```
1278 \cs_new_protected:Npn \@@_pre_array:
1279 {
1280 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1281 \int_gzero_new:N \c@iRow
1282 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1283 \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).

```
1284 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1285 {
1286 \bool_set_true:N \l_@@_last_row_without_value_bool
1287 \bool_if:NT \g_@@_aux_found_bool
1288 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1289 }
1290 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1291 {
1292 \bool_if:NT \g_@@_aux_found_bool
1293 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1294 }
```

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”.

```
1295 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1296 {
1297 \tl_put_right:Nn \@@_update_for_first_and_last_row:
1298 {
```

```

1299         \dim_gset:Nn \g_@@_ht_last_row_dim
1300         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1301         \dim_gset:Nn \g_@@_dp_last_row_dim
1302         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1303     }
1304 }

```

```

1305 \seq_gclear:N \g_@@_cols_vlism_seq
1306 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1307 \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.

```

1308 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1309 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1310 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1311 \@@_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.

```

1312 \box_clear_new:N \l_@@_the_array_box

```

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

```

1313 \@@_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]`).

```

1314 \dim_zero_new:N \l_@@_left_delim_dim
1315 \dim_zero_new:N \l_@@_right_delim_dim
1316 \bool_if:NTF \l_@@_NiceArray_bool
1317 {
1318     \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1319     \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1320 }
1321 {

```

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

```

1322 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1323 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1324 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1325 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1326 }

```

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).

```

1327 \hbox_set:Nw \l_@@_the_array_box

```

```

1328 \skip_horizontal:N \l_@@_left_margin_dim
1329 \skip_horizontal:N \l_@@_extra_left_margin_dim
1330 \c_math_toggle_token
1331 \bool_if:NTF \l_@@_light_syntax_bool
1332 { \use:c { @@-light-syntax } }
1333 { \use:c { @@-normal-syntax } }
1334 }

```

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

```

1335 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1336 {
1337   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1338   \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...

```

1339 \@@_pre_array:
1340 }

```

The `\CodeBefore`

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

```

1341 \cs_new_protected:Npn \@@_pre_code_before:
1342 {

```

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).

```

1343 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1344 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1345 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1346 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }

```

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

```

1347 \pgfsys@markposition { \@@_env: - position }
1348 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1349 \pgfpicture
1350 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1351 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1352 {
1353   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1354   \pgfcoordinate { \@@_env: - row - ##1 }
1355   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1356 }

```

Now, the recreation of the `col` nodes.

```

1357 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1358 {
1359   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1360   \pgfcoordinate { \@@_env: - col - ##1 }
1361   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1362 }

```

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

```

1363 \@@_create_diag_nodes:

```

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

```
1364 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1365 \endpgfpicture
```

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

```
1366 \@@_create_blocks_nodes:
1367 \bool_if:NT \c_@@_tikz_loaded_bool
1368 {
1369   \tikzset
1370   {
1371     every~picture / .style =
1372     { overlay , name~prefix = \@@_env: - }
1373   }
1374 }
1375 \cs_set_eq:NN \cellcolor \@@_cellcolor
1376 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1377 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1378 \cs_set_eq:NN \rowcolor \@@_rowcolor
1379 \cs_set_eq:NN \rowcolors \@@_rowcolors
1380 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1381 \cs_set_eq:NN \arraycolor \@@_arraycolor
1382 \cs_set_eq:NN \columncolor \@@_columncolor
1383 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1384 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1385 }
```

```
1386 \cs_new_protected:Npn \@@_exec_code_before:
1387 {
1388   \seq_gclear_new:N \g_@@_colors_seq
1389   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1390   \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.

```
1391 \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`.

```
1392 \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.

```
1393 \@@_actually_color:
1394 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1395 \group_end:
1396 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1397 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1398 }
```

```
1399 \keys_define:nn { NiceMatrix / CodeBefore }
1400 {
1401   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1402   create-cell-nodes .default:n = true ,
1403   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1404   sub-matrix .value_required:n = true ,
1405   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
```



```

1406     delimiters / color .value_required:n = true ,
1407     unknown .code:n = \@@_error:n { Unknown-key-for~CodeAfter }
1408 }

1409 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1410 {
1411     \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1412     \@@_CodeBefore:w
1413 }

```

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.

```

1414 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1415 {
1416     \bool_if:NT \g_@@_aux_found_bool
1417     {
1418         \@@_pre_code_before:
1419         #1
1420     }
1421 }

```

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.

```

1422 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1423 {
1424     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1425     {
1426         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1427         \pgfcoordinate { \@@_env: - row - ##1 - base }
1428         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1429         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1430         {
1431             \cs_if_exist:cT
1432             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1433             {
1434                 \pgfsys@getposition
1435                 { \@@_env: - ##1 - ####1 - NW }
1436                 \@@_node_position:
1437                 \pgfsys@getposition
1438                 { \@@_env: - ##1 - ####1 - SE }
1439                 \@@_node_position_i:
1440                 \@@_pgf_rect_node:nnn
1441                 { \@@_env: - ##1 - ####1 }
1442                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1443                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1444             }
1445         }
1446     }
1447     \int_step_inline:nn \c@iRow
1448     {
1449         \pgfnodealias
1450         { \@@_env: - ##1 - last }
1451         { \@@_env: - ##1 - \int_use:N \c@jCol }
1452     }
1453     \int_step_inline:nn \c@jCol
1454     {
1455         \pgfnodealias
1456         { \@@_env: - last - ##1 }
1457         { \@@_env: - \int_use:N \c@iRow - ##1 }

```

```

1458     }
1459     \@@_create_extra_nodes:
1460 }

1461 \cs_new_protected:Npn \@@_create_blocks_nodes:
1462 {
1463   \pgfpicture
1464   \pgf@relevantforpicturesizefalse
1465   \pgfrememberpicturepositiononpagetrue
1466   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1467   { \@@_create_one_block_node:nnnnn #1 }
1468   \endpgfpicture
1469 }

```

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

```

1470 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1471 {
1472   \tl_if_empty:nF { #5 }
1473   {
1474     \@@_qpoint:n { col - #2 }
1475     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1476     \@@_qpoint:n { #1 }
1477     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1478     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1479     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1480     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1481     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1482     \@@_pgf_rect_node:nnnnn
1483     { \@@_env: - #5 }
1484     { \dim_use:N \l_tmpa_dim }
1485     { \dim_use:N \l_tmpb_dim }
1486     { \dim_use:N \l_@@_tmpc_dim }
1487     { \dim_use:N \l_@@_tmpd_dim }
1488   }
1489 }

1490 \cs_new_protected:Npn \@@_patch_for_revtext:
1491 {
1492   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1493   \cs_set_eq:NN \insert@column \insert@column@array
1494   \cs_set_eq:NN \@classx \@classx@array
1495   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1496   \cs_set_eq:NN \@arraycr \@arraycr@array
1497   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1498   \cs_set_eq:NN \array \array@array
1499   \cs_set_eq:NN \@array \@array@array
1500   \cs_set_eq:NN \@tabular \@tabular@array
1501   \cs_set_eq:NN \@mkpream \@mkpream@array
1502   \cs_set_eq:NN \endarray \endarray@array
1503   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1504   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1505 }

```

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}

```

1506 \NewDocumentEnvironment { NiceArrayWithDelims }
1507 { m m O { } m ! O { } t \CodeBefore }
1508 {
1509   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1510   \@@_provide_pgfsyspdfmark:
1511   \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.
1512   \bgroup

1513   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1514   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1515   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1516   \int_gzero:N \g_@@_block_box_int
1517   \dim_zero:N \g_@@_width_last_col_dim
1518   \dim_zero:N \g_@@_width_first_col_dim
1519   \bool_gset_false:N \g_@@_row_of_col_done_bool
1520   \str_if_empty:NT \g_@@_name_env_str
1521     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1522   \bool_if:NTF \l_@@_NiceTabular_bool
1523     \mode_leave_vertical:
1524     \@@_test_if_math_mode:
1525   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1526   \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.

```

1527   \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.
1528   \cs_if_exist:NT \tikz@library@external@loaded
1529   {
1530     \tikzexternaldisable
1531     \cs_if_exist:NT \ifstandalone
1532       { \tikzset { external / optimize = false } }
1533   }

```

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

```

1534   \int_gincr:N \g_@@_env_int
1535   \bool_if:NF \l_@@_block_auto_columns_width_bool
1536     { \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`).

```

1537   \seq_gclear:N \g_@@_blocks_seq
1538   \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`.

```

1539   \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1540   \seq_gclear:N \g_@@_pos_of_xdots_seq
1541   \tl_gclear_new:N \g_@@_code_before_tl

```

⁶⁵e.g. `\color[rgb]{0.5,0.5,0}`

```
1542 \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.

```
1543 \bool_gset_false:N \g_@@_aux_found_bool
1544 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1545 {
1546   \bool_gset_true:N \g_@@_aux_found_bool
1547   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1548 }
```

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.

```
1549 \tl_gclear:N \g_@@_aux_tl
1550 \tl_if_empty:NF \g_@@_code_before_tl
1551 {
1552   \bool_set_true:N \l_@@_code_before_bool
1553   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1554 }
```

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.

```
1555 \bool_if:NTF \l_@@_NiceArray_bool
1556 { \keys_set:nn { NiceMatrix / NiceArray } }
1557 { \keys_set:nn { NiceMatrix / pNiceArray } }
1558 { #3 , #5 }

1559 \tl_if_empty:NF \l_@@_rules_color_tl
1560 { \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:.

```
1561 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
1562 }
1563 {
1564   \bool_if:NTF \l_@@_light_syntax_bool
1565   { \use:c { end @@-light-syntax } }
1566   { \use:c { end @@-normal-syntax } }
1567   \c_math_toggle_token
1568   \skip_horizontal:N \l_@@_right_margin_dim
1569   \skip_horizontal:N \l_@@_extra_right_margin_dim
1570   \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.

```
1571 \bool_if:NT \l_@@_width_used_bool
1572 {
1573   \int_compare:nNt \g_@@_total_X_weight_int = 0
1574   { \@@_error:n { width~without~X~columns } }
1575 }
```

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 .

```
1576 \int_compare:nNt \g_@@_total_X_weight_int > 0
1577 {
```

```

1578 \tl_gput_right:Nx \g_@@_aux_tl
1579 {
1580   \bool_set_true:N \l_@@_X_columns_aux_bool
1581   \dim_set:Nn \l_@@_X_columns_dim
1582   {
1583     \dim_compare:nNnTF
1584     {
1585       \dim_abs:n
1586       { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1587     }
1588     <
1589     { 0.001 pt }
1590     { \dim_use:N \l_@@_X_columns_dim }
1591     {
1592       \dim_eval:n
1593       {
1594         ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1595         / \int_use:N \g_@@_total_X_weight_int
1596         + \l_@@_X_columns_dim
1597       }
1598     }
1599   }
1600 }
1601 }

```

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).

```

1602 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1603 {
1604   \bool_if:NF \l_@@_last_row_without_value_bool
1605   {
1606     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1607     {
1608       \@@_error:n { Wrong~last~row }
1609       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1610     }
1611   }
1612 }

```

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

```

1613 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1614 \bool_if:nTF \g_@@_last_col_found_bool
1615 { \int_gdecr:N \c@jCol }
1616 {
1617   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1618   { \@@_error:n { last~col~not~used } }
1619 }

```

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

```

1620 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1621 \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. 129).

```

1622 \int_compare:nNnT \l_@@_first_col_int = 0
1623 {
1624   \skip_horizontal:N \col@sep
1625   \skip_horizontal:N \g_@@_width_first_col_dim

```

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

1626 }

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.

```
1627 \bool_if:NTF \l_@@_NiceArray_bool
1628 {
1629   \str_case:VnF \l_@@_baseline_tl
1630   {
1631     b \@@_use_arraybox_with_notes_b:
1632     c \@@_use_arraybox_with_notes_c:
1633   }
1634   \@@_use_arraybox_with_notes:
1635 }
```

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).

```
1636 {
1637   \int_compare:nNnTF \l_@@_first_row_int = 0
1638   {
1639     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1640     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1641   }
1642   { \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

```
1643 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1644 {
1645   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1646   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1647 }
1648 { \dim_zero:N \l_tmpb_dim }
1649 \hbox_set:Nn \l_tmpa_box
1650 {
1651   \c_math_toggle_token
1652   \tl_if_empty:NF \l_@@_delimiters_color_tl
1653   { \color { \l_@@_delimiters_color_tl } }
1654   \exp_after:wN \left \g_@@_left_delim_tl
1655   \vcenter
1656   {
```

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).

```
1657 \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1658 \hbox
1659 {
1660   \bool_if:NTF \l_@@_NiceTabular_bool
1661   { \skip_horizontal:N -\tabcolsep }
1662   { \skip_horizontal:N -\arraycolsep }
1663   \@@_use_arraybox_with_notes_c:
1664   \bool_if:NTF \l_@@_NiceTabular_bool
1665   { \skip_horizontal:N -\tabcolsep }
1666   { \skip_horizontal:N -\arraycolsep }
1667 }
```

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).

```
1668 \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1669 }
```

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.

```

1670         \tl_if_empty:NF \l_@@_delimiters_color_tl
1671         { \color { \l_@@_delimiters_color_tl } }
1672         \exp_after:wN \right \g_@@_right_delim_tl
1673         \c_math_toggle_token
1674     }

```

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.

```

1675     \bool_if:NTF \l_@@_delimiters_max_width_bool
1676     {
1677         \@@_put_box_in_flow_bis:nn
1678         \g_@@_left_delim_tl \g_@@_right_delim_tl
1679     }
1680     \@@_put_box_in_flow:
1681 }

```

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. 130).

```

1682     \bool_if:NT \g_@@_last_col_found_bool
1683     {
1684         \skip_horizontal:N \g_@@_width_last_col_dim
1685         \skip_horizontal:N \col@sep
1686     }
1687     \bool_if:NF \l_@@_Matrix_bool
1688     {
1689         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1690         { \@@_error:n { columns-not-used } }
1691     }
1692     \group_begin:
1693     \globaldefs = 1
1694     \@@_msg_redirect_name:nn { columns-not-used } { error }
1695     \group_end:
1696     \@@_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.

```

1697     \egroup

```

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

```

1698     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1699     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1700     \iow_now:Nx \@mainaux
1701     {
1702         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _tl }
1703         { \exp_not:V \g_@@_aux_tl }
1704     }
1705     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1706     \bool_if:NT \c_@@_footnote_bool \endsavenotes
1707 }

```

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.

```

1708 \cs_new_protected:Npn \@@_construct_preamble:
1709 {

```

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.

```

1710 \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).

```

1711 \bool_if:NF \l_@@_Matrix_bool
1712 {
1713   \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1714   \@@_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).

```

1715 \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).

```

1716 \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1717 \@tempswatrue

```

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

```

1718 \@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`.

```

1719 \int_gzero:N \c@jCol
1720 \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1721 \bool_gset_false:N \g_tmpb_bool
1722 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1723 {
1724   \tl_gset:Nn \g_@@_preamble_tl
1725   { ! { \skip_horizontal:N \arrayrulewidth } }
1726 }
1727 {
1728   \clist_if_in:NnT \l_@@_vlines_clist 1
1729   {
1730     \tl_gset:Nn \g_@@_preamble_tl
1731     { ! { \skip_horizontal:N \arrayrulewidth } }
1732   }
1733 }

```

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

```

1734 \seq_clear:N \g_@@_cols_vlism_seq

```


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

```
1735 \int_zero:N \l_tmpa_int
```

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

```
1736 \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1737 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1738 }
```

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

```
1739 \bool_if:NT \l_@@_colortbl_like_bool
1740 {
1741   \regex_replace_all:NnN
1742   \c_@@_columncolor_regex
1743   { \c { @@_columncolor_preamble } }
1744   \g_@@_preamble_tl
1745 }
```

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

```
1746 \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}`.

```
1747 \bool_lazy_or:nnT
1748 { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1749 { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1750 { \bool_set_false:N \l_@@_NiceArray_bool }
```

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

```
1751 \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).

```
1752 \int_compare:nNnTF \l_@@_first_col_int = 0
1753 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1754 {
1755   \bool_lazy_all:nT
1756   {
1757     \l_@@_NiceArray_bool
1758     { \bool_not_p:n \l_@@_NiceTabular_bool }
1759     { \tl_if_empty_p:N \l_@@_vlines_clist }
1760     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1761   }
1762   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1763 }
1764 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1765 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1766 {
1767   \bool_lazy_all:nT
1768   {
1769     \l_@@_NiceArray_bool
1770     { \bool_not_p:n \l_@@_NiceTabular_bool }
1771     { \tl_if_empty_p:N \l_@@_vlines_clist }
1772     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1773   }
1774   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1775 }
```

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`).

```
1776 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1777 {
1778   \tl_gput_right:Nn \g_@@_preamble_tl
```

```

1779         { > { \@@_error_too_much_cols: } 1 }
1780     }
1781 }

```

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

```

1782 \cs_new_protected:Npn \@@_patch_preamble:n #1
1783 {
1784     \str_case:nnF { #1 }
1785     {
1786         c { \@@_patch_preamble_i:n #1 }
1787         l { \@@_patch_preamble_i:n #1 }
1788         r { \@@_patch_preamble_i:n #1 }
1789         > { \@@_patch_preamble_ii:nn #1 }
1790         ! { \@@_patch_preamble_ii:nn #1 }
1791         @ { \@@_patch_preamble_ii:nn #1 }
1792         | { \@@_patch_preamble_iii:n #1 }
1793         p { \@@_patch_preamble_iv:n #1 }
1794         b { \@@_patch_preamble_iv:n #1 }
1795         m { \@@_patch_preamble_iv:n #1 }
1796         \@@_V: { \@@_patch_preamble_v:n }
1797         V { \@@_patch_preamble_v:n }
1798         \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
1799         \@@_W: { \@@_patch_preamble_vi:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1800         \@@_S: { \@@_patch_preamble_vii:n }
1801         ( { \@@_patch_preamble_viii:nn #1 }
1802         [ { \@@_patch_preamble_viii:nn #1 }
1803         \{ { \@@_patch_preamble_viii:nn #1 }
1804         ) { \@@_patch_preamble_ix:nn #1 }
1805         ] { \@@_patch_preamble_ix:nn #1 }
1806         \} { \@@_patch_preamble_ix:nn #1 }
1807         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}`.

```

1808     \@@_X { \@@_patch_preamble_x:n }
1809     \q_stop { }
1810 }
1811 {
1812     \str_case:e:nnF { #1 }
1813     {
1814         \l_@@_letter_for_dotted_lines_str
1815         { \@@_patch_preamble_xii:n #1 }
1816         \l_@@_letter_vlism_tl
1817         {
1818             \seq_gput_right:Nx \g_@@_cols_vlism_seq
1819             { \int_eval:n { \c@jCol + 1 } }
1820             \tl_gput_right:Nx \g_@@_preamble_tl
1821             { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1822             \@@_patch_preamble:n
1823         }
1824     }

```

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).

```

1825     {
1826         \keys_set_known:nnN { NiceMatrix / ColumnTypes } { #1 } \l_tmpa_tl
1827         \tl_if_empty:NTF \l_tmpa_tl

```

```

1828         \@@_patch_preamble:n
1829         { \@@_fatal:nn { unknown~column~type } { #1 } }
1830     }
1831 }
1832 }

```

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

```

1833 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1834 {
1835     \tl_gput_right:Nn \g_@@_preamble_tl
1836     {
1837         > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
1838         #1
1839         < \@@_cell_end:
1840     }

```

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

```

1841     \int_gincr:N \c@jCol
1842     \@@_patch_preamble_xi:n
1843 }

```

For >, ! and @

```

1844 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1845 {
1846     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1847     \@@_patch_preamble:n
1848 }

```

For |

```

1849 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1850 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1851     \int_incr:N \l_tmpa_int
1852     \@@_patch_preamble_iii_i:n
1853 }
1854 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1855 {
1856     \str_if_eq:nnTF { #1 } |
1857     { \@@_patch_preamble_iii:n | }
1858     {
1859         \tl_gput_right:Nx \g_@@_preamble_tl
1860         {
1861             \exp_not:N !
1862             {
1863                 \skip_horizontal:n
1864                 {

```

Here, the command \dim_eval:n is mandatory.

```

1865                 \dim_eval:n
1866                 {
1867                     \arrayrulewidth * \l_tmpa_int
1868                     + \doublerulesep * ( \l_tmpa_int - 1 )
1869                 }
1870             }
1871         }
1872     }
1873     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1874     {
1875         \@@_vline:n
1876         {

```

```

1877         position = \int_eval:n { \c@jCol + 1 } ,
1878         multiplicity = \int_use:N \l_tmpa_int ,
1879     }

```

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.

```

1880     }
1881     \int_zero:N \l_tmpa_int
1882     \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
1883     \@@_patch_preamble:n #1
1884 }
1885 }
1886 \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.

```

1887 \keys_define:nn { WithArrows / p-column }
1888 {
1889     r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
1890     r .value_forbidden:n = true ,
1891     c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
1892     c .value_forbidden:n = true ,
1893     l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
1894     l .value_forbidden:n = true ,
1895     si .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
1896     si .value_forbidden:n = true ,
1897     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
1898     p .value_forbidden:n = true ,
1899     t .meta:n = p ,
1900     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
1901     m .value_forbidden:n = true ,
1902     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
1903     b .value_forbidden:n = true ,
1904 }

```

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

```

1905 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
1906 {
1907     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

1908     \@@_patch_preamble_iv_i:n
1909 }
1910 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
1911 {
1912     \str_if_eq:nnTF { #1 } { [ ]
1913     { \@@_patch_preamble_iv_ii:w [ ]
1914     { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
1915 }
1916 \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
1917 { \@@_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.

```

1918 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
1919 {

```

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).

```

1920   \str_set:Nn \l_@@_hpos_col_str { j }
1921   \keys_set:nn { WithArrows / p-column } { #1 }
1922   \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
1923 }

```

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

```

1924 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
1925 {
1926   \use:x
1927   {
1928     \@@_patch_preamble_iv_v:nnnnnnnn
1929     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
1930     { \dim_eval:n { #1 } }
1931   }

```

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.

```

1932   \str_if_eq:VnTF \l_@@_hpos_col_str j
1933   { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
1934   {
1935     \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
1936     { \l_@@_hpos_col_str }
1937   }
1938   \str_case:Vn \l_@@_hpos_col_str
1939   {
1940     c { \exp_not:N \centering }
1941     l { \exp_not:N \raggedright }
1942     r { \exp_not:N \raggedleft }
1943   }
1944 }
1945 { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
1946 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
1947 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
1948 { #2 }
1949 {
1950   \str_case:VnF \l_@@_hpos_col_str
1951   {
1952     { j } { c }
1953     { si } { c }
1954   }
1955   { \l_@@_hpos_col_str }
1956 }
1957 }

```

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

```

1958   \int_gincr:N \c@jCol
1959   \@@_patch_preamble_xi:n
1960 }

```

#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*.

```

1961 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
1962 {
1963   \tl_gput_right:Nn \g_@@_preamble_tl
1964   {
1965     > {

```

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.

```

1966       \dim_set:Nn \l_@@_col_width_dim { #2 }
1967       \@@_cell_begin:w
1968       \begin { #7 } [ #1 ] { #2 }

```

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

```

1969       \everypar
1970       {
1971         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
1972         \everypar { }
1973       }

```

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

```

1974       #3

```

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

```

1975       \g_@@_row_style_tl
1976       \arraybackslash
1977       #5
1978     }
1979     #8
1980     < {
1981       #6

```

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

```

1982       \@finalstrut \@arstrutbox
1983       % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
1984       \end { #7 }

```

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

```

1985       #4
1986       \@@_cell_end:
1987     }
1988   }
1989 }

```

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.

```

1990 \cs_new_protected:Npn \@@_center_cell_box:
1991 {

```

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

```

1992   \tl_gput_right:Nn \g_@@_post_action_cell_tl
1993   {
1994     \int_compare:nNnT
1995     { \box_ht:N \l_@@_cell_box }
1996     >
1997     { \box_ht:N \@arstrutbox }

```

```

1998     {
1999         \hbox_set:Nn \l_@@_cell_box
2000         {
2001             \box_move_down:nn
2002             {
2003                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2004                   + \baselineskip ) / 2
2005             }
2006             { \box_use:N \l_@@_cell_box }
2007         }
2008     }
2009 }
2010 }

```

For V (similar to the V of varwidth).

```

2011 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2012 {
2013     \str_if_eq:nnTF { #1 } { [ ] }
2014     { \@@_patch_preamble_v_i:w [ ] }
2015     { \@@_patch_preamble_v_i:w [ ] { #1 } }
2016 }
2017 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2018 { \@@_patch_preamble_v_ii:nn { #1 } }
2019 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2020 {
2021     \str_set:Nn \l_@@_vpos_col_str { p }
2022     \str_set:Nn \l_@@_hpos_col_str { j }
2023     \keys_set:nn { WithArrows / p-column } { #1 }
2024     \bool_if:NTF \c_@@_varwidth_loaded_bool
2025     { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2026     {
2027         \@@_error:n { varwidth~not~loaded }
2028         \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2029     }
2030 }

```

For w and W

```

2031 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2032 {
2033     \tl_gput_right:Nn \g_@@_preamble_tl
2034     {
2035         > {

```

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.

```

2036         \dim_set:Nn \l_@@_col_width_dim { #4 }
2037         \hbox_set:Nw \l_@@_cell_box
2038         \@@_cell_begin:w
2039         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2040     }
2041     c
2042     < {
2043         \@@_cell_end:
2044         #1
2045         \hbox_set_end:
2046         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2047         \@@_adjust_size_box:
2048         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2049     }
2050 }

```

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

```

2051     \int_gincr:N \c@jCol
2052     \@@_patch_preamble_xi:n
2053 }

```

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).

```

2054 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2055 {
2056   \str_if_eq:nnTF { #1 } { [ ]
2057     { \@@_patch_preamble_vii_i:w [ ]
2058       { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2059     }
2060   \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2061     { \@@_patch_preamble_vii_ii:n { #1 } }
2062   \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2063     {

```

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`.

```

2064   \cs_if_exist:NTF \siunitx_cell_begin:w
2065   {
2066     \tl_gput_right:Nn \g_@@_preamble_tl
2067     {
2068       > {
2069         \@@_cell_begin:w
2070         \keys_set:nn { siunitx } { #1 }
2071         \siunitx_cell_begin:w
2072       }
2073       c
2074       < { \siunitx_cell_end: \@@_cell_end: }
2075     }

```

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

```

2076     \int_gincr:N \c@jCol
2077     \@@_patch_preamble_xi:n
2078   }
2079   { \@@_fatal:n { Version-of-siunitx-too-old } }
2080 }

```

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

```

2081 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2082 {
2083   \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.

```

2084   \int_compare:nNnTF \c@jCol = \c_zero_int
2085   {
2086     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2087     {

```

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

```

2088       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2089       \tl_gset:Nn \g_@@_right_delim_tl { . }
2090       \@@_patch_preamble:n #2
2091     }
2092     {
2093       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2094       \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2095     }
2096   }
2097   { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2098 }
2099 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2100 {
2101   \tl_gput_right:Nx \g_@@_internal_code_after_tl
2102   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }

```



```

2103 \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
2104 {
2105     \@@_error:nn { delimiter~after~opening } { #2 }
2106     \@@_patch_preamble:n
2107 }
2108 { \@@_patch_preamble:n #2 }
2109 }

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}).

2110 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2111 {
2112     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2113     \tl_if_in:nnTF { ) ] \} } { #2 }
2114     { \@@_patch_preamble_ix_i:nnn #1 #2 }
2115     {
2116         \tl_if_eq:nnTF { \q_stop } { #2 }
2117         {
2118             \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2119             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2120             {
2121                 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2122                 \tl_gput_right:Nx \g_@@_internal_code_after_tl
2123                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2124                 \@@_patch_preamble:n #2
2125             }
2126         }
2127         {
2128             \tl_if_in:nnT { ( [ \{ } { #2 }
2129             { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2130             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2131             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2132             \@@_patch_preamble:n #2
2133         }
2134     }
2135 }

2136 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2137 {
2138     \tl_if_eq:nnTF { \q_stop } { #3 }
2139     {
2140         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2141         {
2142             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2143             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2144             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2145             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2146         }
2147         {
2148             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2149             \tl_gput_right:Nx \g_@@_internal_code_after_tl
2150             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2151             \@@_error:nn { double~closing~delimiter } { #2 }
2152         }
2153     }
2154     {
2155         \tl_gput_right:Nx \g_@@_internal_code_after_tl
2156         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2157         \@@_error:nn { double~closing~delimiter } { #2 }
2158         \@@_patch_preamble:n #3
2159     }
2160 }

```

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.

```

2161 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2162 {
2163   \str_if_eq:nnTF { #1 } { [ ]
2164     { \@@_patch_preamble_x_i:w [ ]
2165       { \@@_patch_preamble_x_i:w [ ] #1 }
2166     }
2167   \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2168     { \@@_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).

```

2169 \keys_define:nn { WithArrows / X-column }
2170 { 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.

```

2171 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2172 {

```

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).

```

2173   \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).

```

2174   \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.

```

2175   \int_zero_new:N \l_@@_weight_int
2176   \int_set:Nn \l_@@_weight_int { 1 }
2177   \keys_set:known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
2178   \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2179   \int_compare:nNnT \l_@@_weight_int < 0
2180   {
2181     \@@_error:nx { negative-weight } { \int_use:N \l_@@_weight_int }
2182     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2183   }
2184   \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).

```

2185   \bool_if:NTF \l_@@_X_columns_aux_bool
2186   {
2187     \@@_patch_preamble_iv_iv:nn
2188     { \l_@@_weight_int \l_@@_X_columns_dim }
2189     { minipage }
2190   }
2191   {
2192     \tl_gput_right:Nn \g_@@_preamble_tl
2193     {
2194       > {
2195         \@@_cell_begin:w
2196         \bool_set_true:N \l_@@_X_column_bool

```

The following code will nullify the box of the cell.

```

2197     \tl_gput_right:Nn \g_@@_post_action_cell_tl
2198     { \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`.

```

2199         \begin { minipage } { 5 cm } \arraybackslash
2200     }
2201     c
2202     < {
2203         \end { minipage }
2204         \@@_cell_end:
2205     }
2206 }
2207 \int_gincr:N \c@jCol
2208 \@@_patch_preamble_xi:n
2209 }
2210 }
```

```

2211 \cs_new_protected:Npn \@@_patch_preamble_xii:n #1
2212 {
2213     \tl_gput_right:Nn \g_@@_preamble_tl
2214     { ! { \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`.

```

2215     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2216     { \@@_vdottedline:n { \int_use:N \c@jCol } }
2217     \@@_patch_preamble:n
2218 }
```

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.

```

2219 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2220 {
2221     \str_if_eq:nnTF { #1 } { < }
2222     \@@_patch_preamble_xiii:n
2223     {
2224         \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2225         {
2226             \tl_gput_right:Nn \g_@@_preamble_tl
2227             { ! { \skip_horizontal:N \arrayrulewidth } }
2228         }
2229         {
2230             \exp_args:NNx
2231             \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2232             {
2233                 \tl_gput_right:Nn \g_@@_preamble_tl
2234                 { ! { \skip_horizontal:N \arrayrulewidth } }
2235             }
2236         }
2237         \@@_patch_preamble:n { #1 }
2238     }
2239 }
2240 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2241 {
2242     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2243     \@@_patch_preamble_xi:n
2244 }
```

The redefinition of `\multicolumn`

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

```
2245 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2246 {
```

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`.

```
2247 \multispan { #1 }
2248 \begingroup
2249 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
```

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

```
2250 \@temptokena = { #2 }
2251 \@tempswatrue
2252 \@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.

```
2253 \tl_gclear:N \g_@@_preamble_tl
2254 \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
```

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

```
2255 \exp_args:NV \mkpream \g_@@_preamble_tl
2256 \addtopreamble \@empty
2257 \endgroup
```

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

```
2258 \int_compare:nNnT { #1 } > 1
2259 {
2260   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2261   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2262   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2263   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2264   {
2265     {
2266       \int_compare:nNnTF \c@jCol = 0
2267       { \int_eval:n { \c@iRow + 1 } }
2268       { \int_use:N \c@iRow }
2269     } % modified 2022/01/10
2270     { \int_eval:n { \c@jCol + 1 } }
2271     {
2272       \int_compare:nNnTF \c@jCol = 0
2273       { \int_eval:n { \c@iRow + 1 } }
2274       { \int_use:N \c@iRow }
2275     } % modified 2022/01/10
2276     { \int_eval:n { \c@jCol + #1 } }
2277     { } % for the name of the block
2278   }
2279 }
```

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

```
2280 \cs_set:Npn \@sharp { #3 }
2281 \@arstrut
2282 \@preamble
2283 \null
```

We add some lines.

```
2284 \int_gadd:Nn \c@jCol { #1 - 1 }
2285 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2286 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2287 \ignorespaces
2288 }
```

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`.

```

2289 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2290 {
2291   \str_case:nnF { #1 }
2292   {
2293     c { \@@_patch_m_preamble_i:n #1 }
2294     l { \@@_patch_m_preamble_i:n #1 }
2295     r { \@@_patch_m_preamble_i:n #1 }
2296     > { \@@_patch_m_preamble_ii:nn #1 }
2297     ! { \@@_patch_m_preamble_ii:nn #1 }
2298     @ { \@@_patch_m_preamble_ii:nn #1 }
2299     | { \@@_patch_m_preamble_iii:n #1 }
2300     p { \@@_patch_m_preamble_iv:nnn t #1 }
2301     m { \@@_patch_m_preamble_iv:nnn c #1 }
2302     b { \@@_patch_m_preamble_iv:nnn b #1 }
2303     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2304     \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2305     \q_stop { }
2306   }
2307   { \@@_fatal:nn { unknown~column~type } { #1 } }
2308 }

```

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

```

2309 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2310 {
2311   \tl_gput_right:Nn \g_@@_preamble_tl
2312   {
2313     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2314     #1
2315     < \@@_cell_end:
2316   }

```

We test for the presence of a `<`.

```

2317   \@@_patch_m_preamble_x:n
2318 }

```

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

```

2319 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2320 {
2321   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2322   \@@_patch_m_preamble:n
2323 }

```

For `|`

```

2324 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2325 {
2326   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2327   \@@_patch_m_preamble:n
2328 }

```

For `p`, `m` and `b`

```

2329 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2330 {
2331   \tl_gput_right:Nn \g_@@_preamble_tl
2332   {
2333     > {
2334       \@@_cell_begin:w
2335       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2336       \mode_leave_vertical:
2337       \arraybackslash
2338       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2339     }

```

```

2340     c
2341     < {
2342         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2343         \end { minipage }
2344         \@@_cell_end:
2345     }
2346 }

```

We test for the presence of a <.

```

2347     \@@_patch_m_preamble_x:n
2348 }

```

For w and W

```

2349 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2350 {
2351     \tl_gput_right:Nn \g_@@_preamble_tl
2352     {
2353         > {
2354             \hbox_set:Nw \l_@@_cell_box
2355             \@@_cell_begin:w
2356             \str_set:Nn \l_@@_hpos_cell_str { #3 }
2357         }
2358         c
2359         < {
2360             \@@_cell_end:
2361             #1
2362             \hbox_set_end:
2363             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2364             \@@_adjust_size_box:
2365             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2366         }
2367     }

```

We test for the presence of a <.

```

2368     \@@_patch_m_preamble_x:n
2369 }

```

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.

```

2370 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2371 {
2372     \str_if_eq:nnTF { #1 } { < }
2373     \@@_patch_m_preamble_ix:n
2374     {
2375         \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2376         {
2377             \tl_gput_right:Nn \g_@@_preamble_tl
2378             { ! { \skip_horizontal:N \arrayrulewidth } }
2379         }
2380         {
2381             \exp_args:NNx
2382             \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2383             {
2384                 \tl_gput_right:Nn \g_@@_preamble_tl
2385                 { ! { \skip_horizontal:N \arrayrulewidth } }
2386             }
2387         }
2388         \@@_patch_m_preamble:n { #1 }
2389     }
2390 }
2391 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2392 {
2393     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2394     \@@_patch_m_preamble_x:n
2395 }

```

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`).

```

2396 \cs_new_protected:Npn \@@_put_box_in_flow:
2397 {
2398   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2399   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2400   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2401     { \box_use_drop:N \l_tmpa_box }
2402     \@@_put_box_in_flow_i:
2403 }

```

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).

```

2404 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2405 {
2406   \pgfpicture
2407     \@@_qpoint:n { row - 1 }
2408     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2409     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2410     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2411     \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).

```

2412   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2413     {
2414       \int_set:Nn \l_tmpa_int
2415       {
2416         \str_range:Nnn
2417           \l_@@_baseline_tl
2418           6
2419           { \tl_count:V \l_@@_baseline_tl }
2420       }
2421       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2422     }
2423     {
2424       \str_case:VnF \l_@@_baseline_tl
2425       {
2426         { t } { \int_set:Nn \l_tmpa_int 1 }
2427         { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2428       }
2429       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2430       \bool_lazy_or:nnT
2431         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2432         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2433       {
2434         \@@_error:n { bad~value~for~baseline }
2435         \int_set:Nn \l_tmpa_int 1
2436       }
2437       \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2438     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2439   }
2440   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2441   \endpgfpicture
2442   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2443   \box_use_drop:N \l_tmpa_box
2444 }

```

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).

```
2445 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2446 {
```

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.

```
2447 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2448 {
2449   \box_set_wd:Nn \l_@@_the_array_box
2450   { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2451 }
```

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

```
2452 \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.

```
2453 \hbox
2454 {
2455   \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.

```
2456 \@@_create_extra_nodes:
2457 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2458 }
2459 \bool_lazy_or:nnT
2460 { \int_compare_p:nNn \c@tabularnote > 0 }
2461 { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2462 \@@_insert_tabularnotes:
2463 \end { minipage }
2464 }
2465 \cs_new_protected:Npn \@@_insert_tabularnotes:
2466 {
2467   \skip_vertical:N 0.65ex
```

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

```
2468 \group_begin:
2469 \l_@@_notes_code_before_tl
2470 \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.

```
2471 \int_compare:nNnT \c@tabularnote > 0
2472 {
2473   \bool_if:NTF \l_@@_notes_para_bool
2474   {
2475     \begin { tabularnotes* }
2476     \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2477     \end { tabularnotes* }
```

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

```
2478 \par
2479 }
2480 {
2481   \tabularnotes
2482   \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
```



```

2483         \endtabularnotes
2484     }
2485 }
2486 \unskip
2487 \group_end:
2488 \bool_if:NT \l_@@_notes_bottomrule_bool
2489 {
2490     \bool_if:NTF \c_@@_booktabs_loaded_bool
2491     {

```

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

```

2492         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2493     { \CT@arc@ \hrule height \heavyrulewidth }
2494 }
2495 { \@@_error:n { bottomrule~without~booktabs } }
2496 }
2497 \l_@@_notes_code_after_tl
2498 \seq_gclear:N \g_@@_tabularnotes_seq
2499 \int_gzero:N \c@tabularnote
2500 }

```

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`.

```

2501 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2502 {
2503     \pgfpicture
2504     \@@_qpoint:n { row - 1 }
2505     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2506     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2507     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2508     \endpgfpicture
2509     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2510     \int_compare:nNnT \l_@@_first_row_int = 0
2511     {
2512         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2513         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2514     }
2515     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2516 }

```

Now, the general case.

```

2517 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2518 {

```

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

```

2519     \tl_if_eq:NnT \l_@@_baseline_tl { t }
2520     { \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`.

```

2521     \pgfpicture
2522     \@@_qpoint:n { row - 1 }
2523     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2524     \str_if_in:NnTF \l_@@_baseline_tl { line- }
2525     {
2526         \int_set:Nn \l_tmpa_int
2527         {
2528             \str_range:Nnn
2529             \l_@@_baseline_tl
2530             6
2531             { \tl_count:V \l_@@_baseline_tl }
2532         }

```

```

2533 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2534 }
2535 {
2536 \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2537 \bool_lazy_or:nnT
2538 { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2539 { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2540 {
2541 \@@_error:n { bad-value-for-baseline }
2542 \int_set:Nn \l_tmpa_int 1
2543 }
2544 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2545 }
2546 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2547 \endpgfpicture
2548 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2549 \int_compare:nNnT \l_@@_first_row_int = 0
2550 {
2551 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2552 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2553 }
2554 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2555 }

```

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.

```

2556 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2557 {

```

We will compute the real width of both delimiters used.

```

2558 \dim_zero_new:N \l_@@_real_left_delim_dim
2559 \dim_zero_new:N \l_@@_real_right_delim_dim
2560 \hbox_set:Nn \l_tmpb_box
2561 {
2562 \c_math_toggle_token
2563 \left #1
2564 \vcenter
2565 {
2566 \vbox_to_ht:nn
2567 { \box_ht_plus_dp:N \l_tmpa_box }
2568 { }
2569 }
2570 \right .
2571 \c_math_toggle_token
2572 }
2573 \dim_set:Nn \l_@@_real_left_delim_dim
2574 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }
2575 \hbox_set:Nn \l_tmpb_box
2576 {
2577 \c_math_toggle_token
2578 \left .
2579 \vbox_to_ht:nn
2580 { \box_ht_plus_dp:N \l_tmpa_box }
2581 { }
2582 \right #2
2583 \c_math_toggle_token
2584 }
2585 \dim_set:Nn \l_@@_real_right_delim_dim
2586 { \box_wd:N \l_tmpb_box - \nullldelimiterspace }

```

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

```

2587 \skip_horizontal:N \l_@@_left_delim_dim
2588 \skip_horizontal:N -\l_@@_real_left_delim_dim

```

```

2589 \@@_put_box_in_flow:
2590 \skip_horizontal:N \l_@@_right_delim_dim
2591 \skip_horizontal:N -\l_@@_real_right_delim_dim
2592 }

```

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).

```

2593 \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).

```

2594 {
2595   \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`).

```

2596   { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2597 }
2598 {
2599   \@@_create_col_nodes:
2600   \endarray
2601 }

```

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`).

```

2602 \NewDocumentEnvironment { @@-light-syntax } { b }
2603 {

```

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`.

```

2604   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
2605   \tl_map_inline:nn { #1 }
2606   {
2607     \str_if_eq:nnT { ##1 } { & }
2608     { \@@_fatal:n { ampersand-in~light-syntax } }
2609     \str_if_eq:nnT { ##1 } { \ }
2610     { \@@_fatal:n { double-backslash-in~light-syntax } }
2611   }

```

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`.

```

2612   \@@_light_syntax_i #1 \CodeAfter \q_stop
2613 }

```

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`.

```

2614 { }
2615 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2616 {
2617   \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).

```

2618   \seq_gclear_new:N \g_@@_rows_seq
2619   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2620   \seq_gset_split:NVn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

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

```
2621 \int_compare:nNnT \l_@@_last_row_int = { -1 }
2622 { \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`).

```
2623 \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.

```
2624 \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2625 \@@_line_with_light_syntax_i:V \l_tmpa_tl
2626 \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2627 \@@_create_col_nodes:
2628 \endarray
2629 }

2630 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2631 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }

2632 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2633 {
2634   \seq_gclear_new:N \g_@@_cells_seq
2635   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2636   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2637   \l_tmpa_tl
2638   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2639 }
2640 \cs_generate_variant:Nn \@@_line_with_light_syntax_i:n { V }
```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security).

```
2641 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2642 {
2643   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2644   { \@@_fatal:n { empty~environment } }
```

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

```
2645   \end { #2 }
2646 }
```

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).

```
2647 \cs_new:Npn \@@_create_col_nodes:
2648 {
2649   \crrc
2650   \int_compare:nNnT \l_@@_first_col_int = 0
2651   {
2652     \omit
2653     \hbox_overlap_left:n
2654     {
2655       \bool_if:NT \l_@@_code_before_bool
2656       { \pgfsys@markposition { \@@_env: - col - 0 } }
2657       \pgfpicture
2658       \pgfrememberpicturepositiononpagetrue
2659       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
2660       \str_if_empty:NF \l_@@_name_str
2661       { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2662       \endpgfpicture
```

```

2663         \skip_horizontal:N 2\col@sep
2664         \skip_horizontal:N \g_@@_width_first_col_dim
2665     }
2666     &
2667 }
2668 \omit

```

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

```

2669     \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`).

```

2670     \int_compare:nNnTF \l_@@_first_col_int = 0
2671     {
2672         \bool_if:NT \l_@@_code_before_bool
2673         {
2674             \hbox
2675             {
2676                 \skip_horizontal:N -0.5\arrayrulewidth
2677                 \pgfsys@markposition { \@@_env: - col - 1 }
2678                 \skip_horizontal:N 0.5\arrayrulewidth
2679             }
2680         }
2681         \pgfpicture
2682         \pgfrememberpicturerepositiononpagetrue
2683         \pgfcoordinate { \@@_env: - col - 1 }
2684         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2685         \str_if_empty:NF \l_@@_name_str
2686         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2687         \endpgfpicture
2688     }
2689     {
2690         \bool_if:NT \l_@@_code_before_bool
2691         {
2692             \hbox
2693             {
2694                 \skip_horizontal:N 0.5\arrayrulewidth
2695                 \pgfsys@markposition { \@@_env: - col - 1 }
2696                 \skip_horizontal:N -0.5\arrayrulewidth
2697             }
2698         }
2699         \pgfpicture
2700         \pgfrememberpicturerepositiononpagetrue
2701         \pgfcoordinate { \@@_env: - col - 1 }
2702         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
2703         \str_if_empty:NF \l_@@_name_str
2704         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2705         \endpgfpicture
2706     }

```

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.

```

2707     \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2708     \bool_if:NF \l_@@_auto_columns_width_bool
2709     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2710     {
2711         \bool_lazy_and:nnTF
2712         \l_@@_auto_columns_width_bool
2713         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2714         { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }

```

```

2715         { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2716     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2717 }
2718 \skip_horizontal:N \g_tmpa_skip
2719 \hbox
2720 {
2721     \bool_if:NT \l_@@_code_before_bool
2722     {
2723         \hbox
2724         {
2725             \skip_horizontal:N -0.5\arrayrulewidth
2726             \pgfsys@markposition { \@@_env: - col - 2 }
2727             \skip_horizontal:N 0.5\arrayrulewidth
2728         }
2729     }
2730     \pgfpicture
2731     \pgfrememberpicturepositiononpagetrue
2732     \pgfcoordinate { \@@_env: - col - 2 }
2733     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2734     \str_if_empty:NF \l_@@_name_str
2735     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2736     \endpgfpicture
2737 }

```

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.

```

2738 \int_gset:Nn \g_tmpa_int 1
2739 \bool_if:NNTF \g_@@_last_col_found_bool
2740 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2741 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2742 {
2743     &
2744     \omit
2745     \int_gincr:N \g_tmpa_int

```

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

```

2746     \skip_horizontal:N \g_tmpa_skip
2747     \bool_if:NT \l_@@_code_before_bool
2748     {
2749         \hbox
2750         {
2751             \skip_horizontal:N -0.5\arrayrulewidth
2752             \pgfsys@markposition
2753             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2754             \skip_horizontal:N 0.5\arrayrulewidth
2755         }
2756     }

```

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

```

2757     \pgfpicture
2758     \pgfrememberpicturepositiononpagetrue
2759     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2760     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2761     \str_if_empty:NF \l_@@_name_str
2762     {
2763         \pgfnodealias
2764         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
2765         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2766     }
2767     \endpgfpicture
2768 }

2769 &
2770 \omit

```

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

```

2771 \int_compare:nNt \g_@@_col_total_int = 1
2772 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
2773 \skip_horizontal:N \g_tmpa_skip
2774 \int_gincr:N \g_tmpa_int
2775 \bool_lazy_all:nT
2776 {
2777   \l_@@_NiceArray_bool
2778   { \bool_not_p:n \l_@@_NiceTabular_bool }
2779   { \clist_if_empty_p:N \l_@@_vlines_clist }
2780   { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2781   { ! \l_@@_bar_at_end_of_pream_bool }
2782 }
2783 { \skip_horizontal:N -\col@sep }
2784 \bool_if:NT \l_@@_code_before_bool
2785 {
2786   \hbox
2787   {
2788     \skip_horizontal:N -0.5\arrayrulewidth

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

2789 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2790 { \skip_horizontal:N -\arraycolsep }
2791 \pgfsys@markposition
2792 { \@@_env: - col - \int_eval:n {
2793   \g_tmpa_int + 1 } }
2794 \skip_horizontal:N 0.5\arrayrulewidth
2795 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2796 { \skip_horizontal:N \arraycolsep }
2797 }
2798 }
2799 \pgfpicture
2800 \pgfrememberpicturepositiononpagetrue
2801 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2802 {
2803   \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
2804   {
2805     \pgfpoint
2806     { - 0.5 \arrayrulewidth - \arraycolsep }
2807     \c_zero_dim
2808   }
2809   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2810 }
2811 \str_if_empty:NF \l_@@_name_str
2812 {
2813   \pgfnodealias
2814   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
2815   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
2816 }
2817 \endpgfpicture

2818 \bool_if:NT \g_@@_last_col_found_bool
2819 {
2820   \hbox_overlap_right:n
2821   {
2822     \skip_horizontal:N \g_@@_width_last_col_dim
2823     \bool_if:NT \l_@@_code_before_bool
2824     {
2825       \pgfsys@markposition
2826       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }

```

```

2827     }
2828     \pgfpicture
2829     \pgfrememberpicturepositiononpagetrue
2830     \pgfcoordinate
2831     { \l_@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2832     \pgfpointorigin
2833     \str_if_empty:NF \l_@@_name_str
2834     {
2835         \pgfnodealias
2836         {
2837             \l_@@_name_str - col
2838             - \int_eval:n { \g_@@_col_total_int + 1 }
2839         }
2840         { \l_@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
2841     }
2842     \endpgfpicture
2843 }
2844 }
2845 \cr
2846 }

```

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

```

2847 \tl_const:Nn \c_@@_preamble_first_col_tl
2848 {
2849     >
2850     {

```

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).

```

2851     \cs_set_eq:NN \CodeAfter \l_@@_CodeAfter_i:
2852     \bool_gset_true:N \g_@@_after_col_zero_bool
2853     \l_@@_begin_of_row:

```

The contents of the cell is constructed in the box `\l_@@_cell_box` because we have to compute some dimensions of this box.

```

2854     \hbox_set:Nw \l_@@_cell_box
2855     \l_@@_math_toggle_token:
2856     \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”.

```

2857     \bool_lazy_and:nnT
2858     { \int_compare_p:nNn \c@iRow > 0 }
2859     {
2860         \bool_lazy_or_p:nn
2861         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2862         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2863     }
2864     {
2865         \l_@@_code_for_first_col_tl
2866         \xglobal \colorlet { nicematrix-first-col } { . }
2867     }
2868 }

```

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`.

```

2869     l
2870     <
2871     {
2872         \l_@@_math_toggle_token:
2873         \hbox_set_end:
2874         \bool_if:NT \g_@@_rotate_bool \l_@@_rotate_cell_box:
2875         \l_@@_adjust_size_box:
2876         \l_@@_update_for_first_and_last_row:

```


We actualise the width of the “first column” because we will use this width after the construction of the array.

```
2877 \dim_gset:Nn \g_@@_width_first_col_dim
2878 { \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.

```
2879 \hbox_overlap_left:n
2880 {
2881   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2882     \@@_node_for_cell:
2883     { \box_use_drop:N \l_@@_cell_box }
2884     \skip_horizontal:N \l_@@_left_delim_dim
2885     \skip_horizontal:N \l_@@_left_margin_dim
2886     \skip_horizontal:N \l_@@_extra_left_margin_dim
2887   }
2888   \bool_gset_false:N \g_@@_empty_cell_bool
2889   \skip_horizontal:N -2\col@sep
2890 }
2891 }
```

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

```
2892 \tl_const:Nn \c_@@_preamble_last_col_tl
2893 {
2894   >
2895   {
```

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).

```
2896 \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.

```
2897 \bool_gset_true:N \g_@@_last_col_found_bool
2898 \int_gincr:N \c@jCol
2899 \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.

```
2900 \hbox_set:Nw \l_@@_cell_box
2901 \@@_math_toggle_token:
2902 \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”.

```
2903 \int_compare:nNnT \c@iRow > 0
2904 {
2905   \bool_lazy_or:nnT
2906     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2907     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2908   {
2909     \l_@@_code_for_last_col_tl
2910     \xglobal \colorlet { nicematrix-last-col } { . }
2911   }
2912 }
2913 }
2914 1
2915 <
2916 {
2917   \@@_math_toggle_token:
2918   \hbox_set_end:
2919   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2920   \@@_adjust_size_box:
2921   \@@_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.

```

2922 \dim_gset:Nn \g_@@_width_last_col_dim
2923 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2924 \skip_horizontal:N -2\col@sep

```

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

```

2925 \hbox_overlap_right:n
2926 {
2927   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2928   {
2929     \skip_horizontal:N \l_@@_right_delim_dim
2930     \skip_horizontal:N \l_@@_right_margin_dim
2931     \skip_horizontal:N \l_@@_extra_right_margin_dim
2932     \@@_node_for_cell:
2933   }
2934 }
2935 \bool_gset_false:N \g_@@_empty_cell_bool
2936 }
2937 }

```

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.

```

2938 \NewDocumentEnvironment { NiceArray } { }
2939 {
2940   \bool_set_true:N \l_@@_NiceArray_bool
2941   \str_if_empty:NT \g_@@_name_env_str
2942   { \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).

```

2943 \NiceArrayWithDelims . .
2944 }
2945 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

2946 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2947 {
2948   \NewDocumentEnvironment { #1 NiceArray } { }
2949   {
2950     \str_if_empty:NT \g_@@_name_env_str
2951     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2952     \@@_test_if_math_mode:
2953     \NiceArrayWithDelims #2 #3
2954   }
2955   { \endNiceArrayWithDelims }
2956 }
2957 \@@_def_env:nnn p ( )
2958 \@@_def_env:nnn b [ ]
2959 \@@_def_env:nnn B \{ \}
2960 \@@_def_env:nnn v | |
2961 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2962 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2963 {
2964   \bool_set_true:N \l_@@_Matrix_bool
2965   \use:c { #1 NiceArray }
2966   {
2967     *
2968     {
2969       \int_compare:nNnTF \l_@@_last_col_int < 0
2970       \c@MaxMatrixCols
2971       { \int_eval:n { \l_@@_last_col_int - 1 } }
2972     }
2973     { > \@@_cell_begin:w #2 < \@@_cell_end: }
2974   }
2975 }
2976 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n e }
2977 \clist_map_inline:nn { { } , p , b , B , v , V }
2978 {
2979   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2980   {
2981     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2982     \tl_set:Nn \l_@@_type_of_col_tl c
2983     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2984     \@@_begin_of_NiceMatrix:ne { #1 } \l_@@_type_of_col_tl
2985   }
2986   { \use:c { end #1 NiceArray } }
2987 }

```

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

```

2988 \cs_new_protected:Npn \@@_NotEmpty:
2989 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

2990 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2991 {

```

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.

```

2992   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
2993   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
2994   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2995   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2996   \bool_set_true:N \l_@@_NiceTabular_bool
2997   \NiceArray { #2 }
2998 }
2999 { \endNiceArray }

```

```

3000 \cs_set_protected:Npn \@@_newcolumnntype #1
3001 {
3002   \cs_if_free:cT { NC @ find @ #1 }
3003   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3004   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
3005   \peek_meaning:NTF [
3006     { \newcol@ #1 }
3007     { \newcol@ #1 [ 0 ] }
3008   }

```

```

3009 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3010 {

```

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}`).

```

3011 \bool_if:NT \c_@@_tabularx_loaded_bool { \newcolumnntype { X } { \@@_X } }
3012 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3013 \dim_zero_new:N \l_@@_width_dim
3014 \dim_set:Nn \l_@@_width_dim { #1 }
3015 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3016 \bool_set_true:N \l_@@_NiceTabular_bool
3017 \NiceArray { #3 }
3018 }
3019 { \endNiceArray }

3020 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3021 {
3022   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3023   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3024   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3025   \bool_set_true:N \l_@@_NiceTabular_bool
3026   \NiceArray { #3 }
3027 }
3028 { \endNiceArray }

```

After the construction of the array

```

3029 \cs_new_protected:Npn \@@_after_array:
3030 {
3031   \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.

```

3032 \bool_if:NT \g_@@_last_col_found_bool
3033 { \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`.

```

3034 \bool_if:NT \l_@@_last_col_without_value_bool
3035 { \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.

```

3036 \bool_if:NT \l_@@_last_row_without_value_bool
3037 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3038 \tl_gput_right:Nx \g_@@_aux_tl
3039 {
3040   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3041   {
3042     \int_use:N \l_@@_first_row_int ,
3043     \int_use:N \c_iRow ,
3044     \int_use:N \g_@@_row_total_int ,
3045     \int_use:N \l_@@_first_col_int ,
3046     \int_use:N \c_jCol ,
3047     \int_use:N \g_@@_col_total_int
3048   }
3049 }

```

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`).

```

3050 \seq_if_empty:NF \g_@@_pos_of_blocks_seq

```

```

3051 {
3052   \tl_gput_right:Nx \g_@@_aux_tl
3053   {
3054     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3055     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3056   }
3057 }
3058 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3059 {
3060   \tl_gput_right:Nx \g_@@_aux_tl
3061   {
3062     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3063     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3064     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3065     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3066   }
3067 }

```

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

```

3068   \@@_create_diag_nodes:

```

We create the aliases using `last` for the nodes of the cells in the last row and the last column.

```

3069   \pgfpicture
3070   \int_step_inline:nn \c@iRow
3071   {
3072     \pgfnodealias
3073     { \@@_env: - ##1 - last }
3074     { \@@_env: - ##1 - \int_use:N \c@jCol }
3075   }
3076   \int_step_inline:nn \c@jCol
3077   {
3078     \pgfnodealias
3079     { \@@_env: - last - ##1 }
3080     { \@@_env: - \int_use:N \c@iRow - ##1 }
3081   }
3082   \str_if_empty:NF \l_@@_name_str
3083   {
3084     \int_step_inline:nn \c@iRow
3085     {
3086       \pgfnodealias
3087       { \l_@@_name_str - ##1 - last }
3088       { \@@_env: - ##1 - \int_use:N \c@jCol }
3089     }
3090     \int_step_inline:nn \c@jCol
3091     {
3092       \pgfnodealias
3093       { \l_@@_name_str - last - ##1 }
3094       { \@@_env: - \int_use:N \c@iRow - ##1 }
3095     }
3096   }
3097   \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.

```

3098   \bool_if:NT \l_@@_parallelize_diags_bool
3099   {
3100     \int_gzero_new:N \g_@@_ddots_int
3101     \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots`

⁶⁸It's possible to use the option `parallelize-diags` to disable this parallelization.

diagonals parallel to the first one. Similarly Δ_x and Δ_y of the first Δ diagonal.

```

3102     \dim_gzero_new:N \g_@@_delta_x_one_dim
3103     \dim_gzero_new:N \g_@@_delta_y_one_dim
3104     \dim_gzero_new:N \g_@@_delta_x_two_dim
3105     \dim_gzero_new:N \g_@@_delta_y_two_dim
3106   }
3107   \int_zero_new:N \l_@@_initial_i_int
3108   \int_zero_new:N \l_@@_initial_j_int
3109   \int_zero_new:N \l_@@_final_i_int
3110   \int_zero_new:N \l_@@_final_j_int
3111   \bool_set_false:N \l_@@_initial_open_bool
3112   \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values Δ_x and Δ_y (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.

```

3113   \bool_if:NT \l_@@_small_bool
3114   {
3115     \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
3116     \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }

```

The dimension Δ_x 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.

```

3117     \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
3118   }

```

Now, we actually draw the dotted lines (specified by `\Cdots`, `\Vdots`, etc.).

```

3119   \@@_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.

```

3120   \@@_compute_corners:

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3121   \@@_adjust_pos_of_blocks_seq:
3122   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3123   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

Now, the internal code-after and then, the `\CodeAfter`.

```

3124   \bool_if:NT \c_@@_tikz_loaded_bool
3125   {
3126     \tikzset
3127     {
3128       every~picture / .style =
3129       {
3130         overlay ,
3131         remember~picture ,
3132         name-prefix = \@@_env: -
3133       }
3134     }
3135   }
3136   \cs_set_eq:NN \ialign \@@_old_ialign:
3137   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3138   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3139   \cs_set_eq:NN \OverBrace \@@_OverBrace
3140   \cs_set_eq:NN \line \@@_line
3141   \g_@@_internal_code_after_tl
3142   \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.

```
3143 \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).

```
3144 \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:`.

```
3145 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3146 \scan_stop:
3147 \tl_gclear:N \g_nicematrix_code_after_tl
3148 \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.

```
3149 \tl_if_empty:N \g_nicematrix_code_before_tl
3150 {
```

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`.

```
3151 \cs_set_protected:Npn \rectanglecolor { }
3152 \cs_set_protected:Npn \columncolor { }
3153 \tl_gput_right:Nx \g_@@_aux_tl
3154 {
3155     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3156     { \exp_not:N \g_nicematrix_code_before_tl }
3157 }
3158 \bool_set_true:N \l_@@_code_before_bool
3159 }

3160 \str_gclear:N \g_@@_name_env_str
3161 \@@_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`.

```
3162 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3163 }
```

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`.

```
3164 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3165 { \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).

⁶⁹e.g. `\color[rgb]{0.5,0.5,0}`

```

3166 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3167 {
3168   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3169   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3170 }

```

The following command must *not* be protected.

```

3171 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3172 {
3173   { #1 }
3174   { #2 }
3175   {
3176     \int_compare:nNnTF { #3 } > { 99 }
3177     { \int_use:N \c@iRow }
3178     { #3 }
3179   }
3180   {
3181     \int_compare:nNnTF { #4 } > { 99 }
3182     { \int_use:N \c@jCol }
3183     { #4 }
3184   }
3185   { #5 }
3186 }

```

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).

```

3187 \hook_gput_code:nnn { begindocument } { . }
3188 {
3189   \cs_new_protected:Npx \@@_draw_dotted_lines:
3190   {
3191     \c_@@_pgfortikzpicture_tl
3192     \@@_draw_dotted_lines_i:
3193     \c_@@_endpgfortikzpicture_tl
3194   }
3195 }

```

The following command *must* be protected because it will appear in the construction of the command `\@@_draw_dotted_lines:`.

```

3196 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3197 {
3198   \pgfrememberpicturerepositiononpagetrue
3199   \pgf@relevantforpicturesizefalse
3200   \g_@@_HVdotsfor_lines_tl
3201   \g_@@_Vdots_lines_tl
3202   \g_@@_Ddots_lines_tl
3203   \g_@@_Iddots_lines_tl
3204   \g_@@_Cdots_lines_tl
3205   \g_@@_Ldots_lines_tl
3206 }

3207 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3208 {
3209   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3210   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3211 }

```

We define a new PGF shape for the diag nodes because we want to provide a anchor called `.5` for those nodes.

```

3212 \pgfdeclareshape { @@_diag_node }
3213 {
3214   \savedanchor { \five }
3215   {

```



```

3216         \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3217         \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3218     }
3219     \anchor { 5 } { \five }
3220     \anchor { center } { \pgfpointorigin }
3221 }

```

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).

```

3222 \cs_new_protected:Npn \@@_create_diag_nodes:
3223 {
3224     \pgfpicture
3225     \pgfrememberpicturerepositiononpagetrue
3226     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3227     {
3228         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3229         \dim_set_eq:NN \l_tmpa_dim \pgf@x
3230         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3231         \dim_set_eq:NN \l_tmpb_dim \pgf@y
3232         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3233         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3234         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3235         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3236         \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.

```

3237         \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3238         \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3239         \pgfnode { @_diag_node } { center } { } { \@@_env: - ##1 } { }
3240         \str_if_empty:NF \l_@@_name_str
3241         { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3242     }

```

Now, the last node. Of course, that is only a coordinate because there is not `.5` anchor for that node.

```

3243     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3244     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3245     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3246     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3247     \pgfcoordinate
3248     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3249     \pgfnodealias
3250     { \@@_env: - last }
3251     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3252     \str_if_empty:NF \l_@@_name_str
3253     {
3254         \pgfnodealias
3255         { \l_@@_name_str - \int_use:N \l_tmpa_int }
3256         { \@@_env: - \int_use:N \l_tmpa_int }
3257         \pgfnodealias
3258         { \l_@@_name_str - last }
3259         { \@@_env: - last }
3260     }
3261     \endpgfpicture
3262 }

```

We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on

its right.

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

The command `\l_@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
3263 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3264 {
```

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

```
3265 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3266 \int_set:Nn \l_@@_initial_i_int { #1 }
3267 \int_set:Nn \l_@@_initial_j_int { #2 }
3268 \int_set:Nn \l_@@_final_i_int { #1 }
3269 \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.

```
3270 \bool_set_false:N \l_@@_stop_loop_bool
3271 \bool_do_until:Nn \l_@@_stop_loop_bool
3272 {
3273   \int_add:Nn \l_@@_final_i_int { #3 }
3274   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3275   \bool_set_false:N \l_@@_final_open_bool
3276   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3277   {
3278     \int_compare:nNnTF { #3 } = 1
3279     { \bool_set_true:N \l_@@_final_open_bool }
3280     {
3281       \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3282       { \bool_set_true:N \l_@@_final_open_bool }
3283     }
3284   }
3285   {
3286     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3287     {
3288       \int_compare:nNnTF { #4 } = { -1 }
3289       { \bool_set_true:N \l_@@_final_open_bool }
3290     }
3291     {
```

```

3292         \int_compare:nNtT \l_@@_final_j_int > \l_@@_col_max_int
3293         {
3294             \int_compare:nNtT { #4 } = 1
3295             { \bool_set_true:N \l_@@_final_open_bool }
3296         }
3297     }
3298 }
3299 \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.

```

3300 {

```

We do a step backwards.

```

3301     \int_sub:Nn \l_@@_final_i_int { #3 }
3302     \int_sub:Nn \l_@@_final_j_int { #4 }
3303     \bool_set_true:N \l_@@_stop_loop_bool
3304 }

```

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`.

```

3305 {
3306     \cs_if_exist:cTF
3307     {
3308         @@ _ dotted _
3309         \int_use:N \l_@@_final_i_int -
3310         \int_use:N \l_@@_final_j_int
3311     }
3312     {
3313         \int_sub:Nn \l_@@_final_i_int { #3 }
3314         \int_sub:Nn \l_@@_final_j_int { #4 }
3315         \bool_set_true:N \l_@@_final_open_bool
3316         \bool_set_true:N \l_@@_stop_loop_bool
3317     }
3318     {
3319         \cs_if_exist:cTF
3320         {
3321             pgf @ sh @ ns @ \@@_env:
3322             - \int_use:N \l_@@_final_i_int
3323             - \int_use:N \l_@@_final_j_int
3324         }
3325         { \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.

```

3326     {
3327         \cs_set:cpn
3328         {
3329             @@ _ dotted _
3330             \int_use:N \l_@@_final_i_int -
3331             \int_use:N \l_@@_final_j_int
3332         }
3333         { }
3334     }
3335 }
3336 }
3337 }

```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programming is similar to the previous one.

```

3338     \bool_set_false:N \l_@@_stop_loop_bool

```

```

3339 \bool_do_until:Nn \l_@@_stop_loop_bool
3340 {
3341   \int_sub:Nn \l_@@_initial_i_int { #3 }
3342   \int_sub:Nn \l_@@_initial_j_int { #4 }
3343   \bool_set_false:N \l_@@_initial_open_bool
3344   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3345   {
3346     \int_compare:nNnTF { #3 } = 1
3347     { \bool_set_true:N \l_@@_initial_open_bool }
3348     {
3349       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3350       { \bool_set_true:N \l_@@_initial_open_bool }
3351     }
3352   }
3353   {
3354     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3355     {
3356       \int_compare:nNnT { #4 } = 1
3357       { \bool_set_true:N \l_@@_initial_open_bool }
3358     }
3359     {
3360       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3361       {
3362         \int_compare:nNnT { #4 } = { -1 }
3363         { \bool_set_true:N \l_@@_initial_open_bool }
3364       }
3365     }
3366   }
3367   \bool_if:NnTF \l_@@_initial_open_bool
3368   {
3369     \int_add:Nn \l_@@_initial_i_int { #3 }
3370     \int_add:Nn \l_@@_initial_j_int { #4 }
3371     \bool_set_true:N \l_@@_stop_loop_bool
3372   }
3373   {
3374     \cs_if_exist:cTF
3375     {
3376       @@ _ dotted _
3377       \int_use:N \l_@@_initial_i_int -
3378       \int_use:N \l_@@_initial_j_int
3379     }
3380     {
3381       \int_add:Nn \l_@@_initial_i_int { #3 }
3382       \int_add:Nn \l_@@_initial_j_int { #4 }
3383       \bool_set_true:N \l_@@_initial_open_bool
3384       \bool_set_true:N \l_@@_stop_loop_bool
3385     }
3386     {
3387       \cs_if_exist:cTF
3388       {
3389         pgf @ sh @ ns @ \@@_env:
3390         - \int_use:N \l_@@_initial_i_int
3391         - \int_use:N \l_@@_initial_j_int
3392       }
3393       { \bool_set_true:N \l_@@_stop_loop_bool }
3394       {
3395         \cs_set:cpn
3396         {
3397           @@ _ dotted _
3398           \int_use:N \l_@@_initial_i_int -
3399           \int_use:N \l_@@_initial_j_int
3400         }
3401         { }

```

```

3402         }
3403     }
3404 }
3405 }

```

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.

```

3406 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3407 {
3408     { \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`.

```

3409     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3410     { \int_use:N \l_@@_final_i_int }
3411     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3412     { } % for the name of the block
3413 }
3414 }

```

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).

```

3415 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3416 {
3417     \int_set:Nn \l_@@_row_min_int 1
3418     \int_set:Nn \l_@@_col_min_int 1
3419     \int_set_eq:NN \l_@@_row_max_int \c{iRow}
3420     \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`.

```

3421 \seq_map_inline:Nn \g_@@_submatrix_seq
3422 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
3423 }

```

#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.

```

3424 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3425 {
3426     \bool_if:nT
3427     {
3428         \int_compare_p:n { #3 <= #1 }
3429         && \int_compare_p:n { #1 <= #5 }
3430         && \int_compare_p:n { #4 <= #2 }
3431         && \int_compare_p:n { #2 <= #6 }
3432     }
3433     {
3434         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3435         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3436         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3437         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3438     }
3439 }

```

```

3440 \cs_new_protected:Npn \@@_set_initial_coords:
3441 {
3442     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3443     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3444 }
3445 \cs_new_protected:Npn \@@_set_final_coords:
3446 {

```

```

3447 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3448 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3449 }
3450 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3451 {
3452   \pgfpointanchor
3453   {
3454     \@@_env:
3455     - \int_use:N \l_@@_initial_i_int
3456     - \int_use:N \l_@@_initial_j_int
3457   }
3458   { #1 }
3459   \@@_set_initial_coords:
3460 }
3461 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3462 {
3463   \pgfpointanchor
3464   {
3465     \@@_env:
3466     - \int_use:N \l_@@_final_i_int
3467     - \int_use:N \l_@@_final_j_int
3468   }
3469   { #1 }
3470   \@@_set_final_coords:
3471 }
3472 \cs_new_protected:Npn \@@_open_x_initial_dim:
3473 {
3474   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3475   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3476   {
3477     \cs_if_exist:cT
3478     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3479     {
3480       \pgfpointanchor
3481       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3482       { west }
3483       \dim_set:Nn \l_@@_x_initial_dim
3484       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3485     }
3486   }
3487   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3488   {
3489     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3490     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3491     \dim_add:Nn \l_@@_x_initial_dim \col@sep
3492   }
3493 }
3494 \cs_new_protected:Npn \@@_open_x_final_dim:
3495 {
3496   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3497   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3498   {
3499     \cs_if_exist:cT
3500     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3501     {
3502       \pgfpointanchor
3503       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3504       { east }
3505       \dim_set:Nn \l_@@_x_final_dim
3506       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3507     }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```
3508 }
```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```
3509 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3510 {
3511   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
3512   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3513   \dim_sub:Nn \l_@@_x_final_dim \col@sep
3514 }
3515 }
```

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.

```
3516 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3517 {
3518   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3519   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3520   {
3521     \@@_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.

```
3522 \group_begin:
3523   \int_compare:nNnTF { #1 } = 0
3524   { \color { nicematrix-first-row } }
3525   {
```

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.

```
3526   \int_compare:nNnT { #1 } = \l_@@_last_row_int
3527   { \color { nicematrix-last-row } }
3528   }
3529   \keys_set:nn { NiceMatrix / xdots } { #3 }
3530   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3531   \@@_actually_draw_Ldots:
3532 \group_end:
3533 }
3534 }
```

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`.

```
3535 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3536 {
3537   \bool_if:NTF \l_@@_initial_open_bool
3538   {
3539     \@@_open_x_initial_dim:
3540     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3541     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3542   }
3543   { \@@_set_initial_coords_from_anchor:n { base-east } }
3544   \bool_if:NTF \l_@@_final_open_bool
```

```

3545 {
3546   \@@_open_x_final_dim:
3547   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3548   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3549 }
3550 { \@@_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 (?).

```

3551   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3552   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3553   \@@_draw_line:
3554 }

```

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.

```

3555 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3556 {
3557   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3558   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3559   {
3560     \@@_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.

```

3561   \group_begin:
3562   \int_compare:nNnTF { #1 } = 0
3563   { \color { nicematrix-first-row } }
3564   {

```

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.

```

3565     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3566     { \color { nicematrix-last-row } }
3567   }
3568   \keys_set:nn { NiceMatrix / xdots } { #3 }
3569   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3570   \@@_actually_draw_Cdots:
3571   \group_end:
3572 }
3573 }

```

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`.

```

3574 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3575 {
3576   \bool_if:NTF \l_@@_initial_open_bool
3577   { \@@_open_x_initial_dim: }
3578   { \@@_set_initial_coords_from_anchor:n { mid~east } }
3579   \bool_if:NTF \l_@@_final_open_bool
3580   { \@@_open_x_final_dim: }
3581   { \@@_set_final_coords_from_anchor:n { mid~west } }
3582   \bool_lazy_and:nnTF

```



```

3583 \l_@@_initial_open_bool
3584 \l_@@_final_open_bool
3585 {
3586   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3587   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3588   \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
3589   \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3590   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3591 }
3592 {
3593   \bool_if:NT \l_@@_initial_open_bool
3594     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3595   \bool_if:NT \l_@@_final_open_bool
3596     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3597 }
3598 \@@_draw_line:
3599 }
3600 \cs_new_protected:Npn \@@_open_y_initial_dim:
3601 {
3602   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3603   \dim_set:Nn \l_@@_y_initial_dim
3604     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3605   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3606     {
3607       \cs_if_exist:cT
3608         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3609         {
3610           \pgfpointanchor
3611             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3612             { north }
3613           \dim_set:Nn \l_@@_y_initial_dim
3614             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3615         }
3616     }
3617 }
3618 \cs_new_protected:Npn \@@_open_y_final_dim:
3619 {
3620   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3621   \dim_set:Nn \l_@@_y_final_dim
3622     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3623   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3624     {
3625       \cs_if_exist:cT
3626         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3627         {
3628           \pgfpointanchor
3629             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3630             { south }
3631           \dim_set:Nn \l_@@_y_final_dim
3632             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3633         }
3634     }
3635 }

```

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.

```

3636 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3637 {
3638   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3639   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3640   {
3641     \@@_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.

```

3642     \group_begin:
3643     \int_compare:nNnTF { #2 } = 0
3644     { \color { nicematrix-first-col } }
3645     {
3646         \int_compare:nNnT { #2 } = \l_@@_last_col_int
3647         { \color { nicematrix-last-col } }
3648     }
3649     \keys_set:nn { NiceMatrix / xdots } { #3 }
3650     \tl_if_empty:VF \l_@@_xdots_color_tl
3651     { \color { \l_@@_xdots_color_tl } }
3652     \@@_actually_draw_Vdots:
3653 \group_end:
3654 }
3655 }
```

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`.

```

3656 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3657 {
```

The boolean `\l_tmpa_bool` indicates whether the column is of type `l` or may be considered as if.

```

3658     \bool_set_false:N \l_tmpa_bool
```

First the case when the line is closed on both ends.

```

3659     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3660     {
3661         \@@_set_initial_coords_from_anchor:n { south-west }
3662         \@@_set_final_coords_from_anchor:n { north-west }
3663         \bool_set:Nn \l_tmpa_bool
3664         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3665     }
```

Now, we try to determine whether the column is of type `c` or may be considered as if.

```

3666     \bool_if:NTF \l_@@_initial_open_bool
3667     \@@_open_y_initial_dim:
3668     { \@@_set_initial_coords_from_anchor:n { south } }
3669     \bool_if:NTF \l_@@_final_open_bool
3670     \@@_open_y_final_dim:
3671     { \@@_set_final_coords_from_anchor:n { north } }
3672     \bool_if:NTF \l_@@_initial_open_bool
3673     {
3674         \bool_if:NTF \l_@@_final_open_bool
3675         {
3676             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3677             \dim_set_eq:NN \l_tmpa_dim \pgf@x
3678             \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
3679             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3680             \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.

```

3681         \int_compare:nNnT \l_@@_last_col_int > { -2 }
3682         {
3683             \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3684             {
3685                 \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3686                 \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3687                 \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3688                 \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3689             }
3690         }
3691     }
3692     { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3693 }
3694 {
3695     \bool_if:NTF \l_@@_final_open_bool
3696     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3697     {

```

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.

```

3698         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3699         {
3700             \dim_set:Nn \l_@@_x_initial_dim
3701             {
3702                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3703                 \l_@@_x_initial_dim \l_@@_x_final_dim
3704             }
3705             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3706         }
3707     }
3708 }
3709 \@@_draw_line:
3710 }

```

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.

```

3711 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3712 {
3713     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3714     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3715     {
3716         \@@_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.

```

3717     \group_begin:
3718     \keys_set:nn { NiceMatrix / xdots } { #3 }
3719     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3720     \@@_actually_draw_Ddots:
3721     \group_end:
3722 }
3723 }

```

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`.

```

3724 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3725 {
3726   \bool_if:NTF \l_@@_initial_open_bool
3727   {
3728     \@@_open_y_initial_dim:
3729     \@@_open_x_initial_dim:
3730   }
3731   { \@@_set_initial_coords_from_anchor:n { south-east } }
3732   \bool_if:NTF \l_@@_final_open_bool
3733   {
3734     \@@_open_x_final_dim:
3735     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3736   }
3737   { \@@_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.

```

3738   \bool_if:NT \l_@@_parallelize_diags_bool
3739   {
3740     \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).

```

3741     \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.

```

3742     {
3743       \dim_gset:Nn \g_@@_delta_x_one_dim
3744       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3745       \dim_gset:Nn \g_@@_delta_y_one_dim
3746       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3747     }

```

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`.

```

3748     {
3749       \dim_set:Nn \l_@@_y_final_dim
3750       {
3751         \l_@@_y_initial_dim +
3752         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3753         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3754       }
3755     }
3756   }
3757   \@@_draw_line:
3758 }

```

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.

```

3759 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3760 {
3761   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3762   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }

```

```

3763 {
3764   \@@_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.

```

3765   \group_begin:
3766     \keys_set:nn { NiceMatrix / xdots } { #3 }
3767     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3768     \@@_actually_draw_Iddots:
3769   \group_end:
3770 }
3771 }

```

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.`

```

3772 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3773 {
3774   \bool_if:NTF \l_@@_initial_open_bool
3775   {
3776     \@@_open_y_initial_dim:
3777     \@@_open_x_initial_dim:
3778   }
3779   { \@@_set_initial_coords_from_anchor:n { south-west } }
3780   \bool_if:NTF \l_@@_final_open_bool
3781   {
3782     \@@_open_y_final_dim:
3783     \@@_open_x_final_dim:
3784   }
3785   { \@@_set_final_coords_from_anchor:n { north-east } }
3786   \bool_if:NT \l_@@_parallelize_diags_bool
3787   {
3788     \int_gincr:N \g_@@_iddots_int
3789     \int_compare:nNnTF \g_@@_iddots_int = 1
3790     {
3791       \dim_gset:Nn \g_@@_delta_x_two_dim
3792       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3793       \dim_gset:Nn \g_@@_delta_y_two_dim
3794       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3795     }
3796     {
3797       \dim_set:Nn \l_@@_y_final_dim
3798       {
3799         \l_@@_y_initial_dim +
3800         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3801         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3802       }
3803     }
3804   }
3805   \@@_draw_line:
3806 }

```

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`

```

3807 \cs_new_protected:Npn \@@_draw_line:
3808 {
3809   \pgfrememberpicturepositiononpagetrue
3810   \pgf@relevantforpicturesizefalse
3811   \bool_lazy_or:nnTF
3812   { \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.

```

3813   \l_@@_dotted_bool
3814   \@@_draw_standard_dotted_line:
3815   \@@_draw_unstandard_dotted_line:
3816 }

```

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.

```

3817 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
3818 {
3819   \begin { scope }
3820   \@@_draw_unstandard_dotted_line:o
3821   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3822 }

```

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.

```

3823 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
3824 {
3825   \@@_draw_unstandard_dotted_line:nVV
3826   { #1 }
3827   \l_@@_xdots_up_tl
3828   \l_@@_xdots_down_tl
3829 }
3830 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
3831 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
3832 {
3833   \draw
3834   [
3835     #1 ,
3836     shorten-> = \l_@@_xdots_shorten_dim ,
3837     shorten-< = \l_@@_xdots_shorten_dim ,
3838   ]
3839   ( \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).

```

3840   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3841   node [ sloped , below ] { $ \scriptstyle #3 $ }
3842   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;

```

```

3843   \end { scope }
3844 }
3845 \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).

```

3846 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3847 {
3848   \bool_lazy_and:nnF
3849   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3850   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3851   {
3852     \pgfscope
3853     \pgftransformshift
3854     {
3855       \pgfpointlineattime { 0.5 }
3856       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3857       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3858     }
3859     \pgftransformrotate
3860     {
3861       \fp_eval:n
3862       {
3863         atand
3864         (
3865           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3866           \l_@@_x_final_dim - \l_@@_x_initial_dim
3867         )
3868       }
3869     }
3870     \pgfnode
3871     { rectangle }
3872     { south }
3873     {
3874       \c_math_toggle_token
3875       \scriptstyle \l_@@_xdots_up_tl
3876       \c_math_toggle_token
3877     }
3878     { }
3879     { \pgfusepath { } }
3880     \pgfnode
3881     { rectangle }
3882     { north }
3883     {
3884       \c_math_toggle_token
3885       \scriptstyle \l_@@_xdots_down_tl
3886       \c_math_toggle_token
3887     }
3888     { }
3889     { \pgfusepath { } }
3890     \endpgfscope
3891   }
3892   \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.

```

3893   \dim_zero_new:N \l_@@_l_dim
3894   \dim_set:Nn \l_@@_l_dim
3895   {
3896     \fp_to_dim:n
3897     {
3898       sqrt
3899       (

```

```

3900          ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3901          +
3902          ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3903      )
3904  }
3905  }

```

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.

```

3906      \bool_lazy_or:nnF
3907      { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3908      { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3909      \@@_draw_standard_dotted_line_i:
3910  \group_end:
3911  }
3912  \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3913  \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3914  {

```

The number of dots will be `\l_tmpa_int + 1`.

```

3915      \bool_if:NTF \l_@@_initial_open_bool
3916      {
3917          \bool_if:NTF \l_@@_final_open_bool
3918          {
3919              \int_set:Nn \l_tmpa_int
3920              { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3921          }
3922          {
3923              \int_set:Nn \l_tmpa_int
3924              {
3925                  \dim_ratio:nn
3926                  { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3927                  \l_@@_inter_dots_dim
3928              }
3929          }
3930      }
3931      {
3932          \bool_if:NTF \l_@@_final_open_bool
3933          {
3934              \int_set:Nn \l_tmpa_int
3935              {
3936                  \dim_ratio:nn
3937                  { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3938                  \l_@@_inter_dots_dim
3939              }
3940          }
3941          {
3942              \int_set:Nn \l_tmpa_int
3943              {
3944                  \dim_ratio:nn
3945                  { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3946                  \l_@@_inter_dots_dim
3947              }
3948          }
3949      }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

3950      \dim_set:Nn \l_tmpa_dim
3951      {
3952          ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *

```



```

3953     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3954   }
3955   \dim_set:Nn \l_tmpb_dim
3956   {
3957     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3958     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3959   }

```

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`.

```

3960   \int_set:Nn \l_tmpb_int
3961   {
3962     \bool_if:NTF \l_@@_initial_open_bool
3963     { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3964     { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3965   }

```

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.

```

3966   \dim_gadd:Nn \l_@@_x_initial_dim
3967   {
3968     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3969     \dim_ratio:nn
3970     { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3971     { 2 \l_@@_l_dim }
3972     * \l_tmpb_int
3973   }
3974   \dim_gadd:Nn \l_@@_y_initial_dim
3975   {
3976     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3977     \dim_ratio:nn
3978     { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3979     { 2 \l_@@_l_dim }
3980     * \l_tmpb_int
3981   }
3982   \pgf@relevantforpicturesizefalse
3983   \int_step_inline:nnn 0 \l_tmpa_int
3984   {
3985     \pgfpathcircle
3986     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3987     { \l_@@_radius_dim }
3988     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3989     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3990   }
3991   \pgfusepathqfill
3992 }

```

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.

```

3993 \hook_gput_code:nnn { begindocument } { . }
3994 {
3995   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3996   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3997   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3998   {
3999     \int_compare:nNnTF \c@jCol = 0
4000     { \@@_error:nn { in~first~col } \Ldots }
4001     {
4002       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4003       { \@@_error:nn { in~last~col } \Ldots }
4004       {
4005         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4006         { #1 , down = #2 , up = #3 }
4007       }
4008     }
4009     \bool_if:NF \l_@@_nullify_dots_bool
4010     { \phantom { \ensuremath { \@@_old_ldots } } }
4011     \bool_gset_true:N \g_@@_empty_cell_bool
4012   }

4013   \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
4014   {
4015     \int_compare:nNnTF \c@jCol = 0
4016     { \@@_error:nn { in~first~col } \Cdots }
4017     {
4018       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4019       { \@@_error:nn { in~last~col } \Cdots }
4020       {
4021         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4022         { #1 , down = #2 , up = #3 }
4023       }
4024     }
4025     \bool_if:NF \l_@@_nullify_dots_bool
4026     { \phantom { \ensuremath { \@@_old_cdots } } }
4027     \bool_gset_true:N \g_@@_empty_cell_bool
4028   }

4029   \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4030   {
4031     \int_compare:nNnTF \c@iRow = 0
4032     { \@@_error:nn { in~first~row } \Vdots }
4033     {
4034       \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4035       { \@@_error:nn { in~last~row } \Vdots }
4036       {
4037         \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4038         { #1 , down = #2 , up = #3 }
4039       }
4040     }
4041     \bool_if:NF \l_@@_nullify_dots_bool
4042     { \phantom { \ensuremath { \@@_old_vdots } } }
4043     \bool_gset_true:N \g_@@_empty_cell_bool
4044   }

4045   \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4046   {
4047     \int_case:nnF \c@iRow
4048     {
4049       0 { \@@_error:nn { in~first~row } \Ddots }
4050       \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }

```

```

4051     }
4052     {
4053         \int_case:nnF \c@jCol
4054         {
4055             0 { \@@_error:nn { in~first~col } \Ddots }
4056             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4057         }
4058         {
4059             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4060             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4061             { #1 , down = #2 , up = #3 }
4062         }
4063     }
4064 }
4065 \bool_if:NF \l_@@_nullify_dots_bool
4066 { \phantom { \ensuremath { \@@_old_ddots } } }
4067 \bool_gset_true:N \g_@@_empty_cell_bool
4068 }

4069 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4070 {
4071     \int_case:nnF \c@iRow
4072     {
4073         0 { \@@_error:nn { in~first~row } \Iddots }
4074         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4075     }
4076     {
4077         \int_case:nnF \c@jCol
4078         {
4079             0 { \@@_error:nn { in~first~col } \Iddots }
4080             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4081         }
4082         {
4083             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4084             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4085             { #1 , down = #2 , up = #3 }
4086         }
4087     }
4088     \bool_if:NF \l_@@_nullify_dots_bool
4089     { \phantom { \ensuremath { \@@_old_iddots } } }
4090     \bool_gset_true:N \g_@@_empty_cell_bool
4091 }
4092 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

4093 \keys_define:nn { NiceMatrix / Ddots }
4094 {
4095     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4096     draw-first .default:n = true ,
4097     draw-first .value_forbidden:n = true
4098 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

4099 \cs_new_protected:Npn \@@_Hspace:
4100 {
4101     \bool_gset_true:N \g_@@_empty_cell_bool
4102     \hspace
4103 }

```

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`.

```
4104 \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`.

```
4105 \cs_new:Npn \@@_Hdotsfor:
4106 {
4107   \bool_lazy_and:nnTF
4108     { \int_compare_p:nNn \c@jCol = 0 }
4109     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4110     {
4111       \bool_if:NTF \g_@@_after_col_zero_bool
4112       {
4113         \multicolumn { 1 } { c } { }
4114         \@@_Hdotsfor_i
4115       }
4116       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4117     }
4118   {
4119     \multicolumn { 1 } { c } { }
4120     \@@_Hdotsfor_i
4121   }
4122 }
```

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:`).

```
4123 \hook_gput_code:nnn { begindocument } { . }
4124 {
4125   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4126   \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.

```
4127   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4128   {
4129     \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4130     {
4131       \@@_Hdotsfor:nnnn
4132       { \int_use:N \c@iRow }
4133       { \int_use:N \c@jCol }
4134       { #2 }
4135       {
4136         #1 , #3 ,
4137         down = \exp_not:n { #4 } ,
4138         up = \exp_not:n { #5 }
4139       }
4140     }
4141     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4142   }
4143 }
```

Enf of `\AddToHook`.

```
4144 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4145 {
4146   \bool_set_false:N \l_@@_initial_open_bool
4147   \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
4148   \int_set:Nn \l_@@_initial_i_int { #1 }
4149   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

For the column, it's a bit more complicated.

```

4150 \int_compare:nNnTF { #2 } = 1
4151 {
4152   \int_set:Nn \l_@@_initial_j_int 1
4153   \bool_set_true:N \l_@@_initial_open_bool
4154 }
4155 {
4156   \cs_if_exist:cTF
4157   {
4158     pgf @ sh @ ns @ \@@_env:
4159     - \int_use:N \l_@@_initial_i_int
4160     - \int_eval:n { #2 - 1 }
4161   }
4162   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4163   {
4164     \int_set:Nn \l_@@_initial_j_int { #2 }
4165     \bool_set_true:N \l_@@_initial_open_bool
4166   }
4167 }
4168 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4169 {
4170   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4171   \bool_set_true:N \l_@@_final_open_bool
4172 }
4173 {
4174   \cs_if_exist:cTF
4175   {
4176     pgf @ sh @ ns @ \@@_env:
4177     - \int_use:N \l_@@_final_i_int
4178     - \int_eval:n { #2 + #3 }
4179   }
4180   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4181   {
4182     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4183     \bool_set_true:N \l_@@_final_open_bool
4184   }
4185 }
4186 \group_begin:
4187 \int_compare:nNnTF { #1 } = 0
4188 { \color { nicematrix-first-row } }
4189 {
4190   \int_compare:nNnT { #1 } = \g_@@_row_total_int
4191   { \color { nicematrix-last-row } }
4192 }
4193 \keys_set:nn { NiceMatrix / xdots } { #4 }
4194 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4195 \@@_actually_draw_Ldots:
4196 \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).

```

4197 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4198 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4199 }

4200 \hook_gput_code:nnn { begindocument } { . }
4201 {
4202   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4203   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4204   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4205   {

```

```

4206 \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4207 {
4208   \@@_Vdotsfor:nmmn
4209   { \int_use:N \c@iRow }
4210   { \int_use:N \c@jCol }
4211   { #2 }
4212   {
4213     #1 , #3 ,
4214     down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4215   }
4216 }
4217 }
4218 }

```

Enf of \AddToHook.

```

4219 \cs_new_protected:Npn \@@_Vdotsfor:nmmn #1 #2 #3 #4
4220 {
4221   \bool_set_false:N \l_@@_initial_open_bool
4222   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

4223 \int_set:Nn \l_@@_initial_j_int { #2 }
4224 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

4225 \int_compare:nNnTF #1 = 1
4226 {
4227   \int_set:Nn \l_@@_initial_i_int 1
4228   \bool_set_true:N \l_@@_initial_open_bool
4229 }
4230 {
4231   \cs_if_exist:cTF
4232   {
4233     pgf @ sh @ ns @ \@@_env:
4234     - \int_eval:n { #1 - 1 }
4235     - \int_use:N \l_@@_initial_j_int
4236   }
4237   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4238   {
4239     \int_set:Nn \l_@@_initial_i_int { #1 }
4240     \bool_set_true:N \l_@@_initial_open_bool
4241   }
4242 }
4243 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4244 {
4245   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4246   \bool_set_true:N \l_@@_final_open_bool
4247 }
4248 {
4249   \cs_if_exist:cTF
4250   {
4251     pgf @ sh @ ns @ \@@_env:
4252     - \int_eval:n { #1 + #3 }
4253     - \int_use:N \l_@@_final_j_int
4254   }
4255   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4256   {
4257     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4258     \bool_set_true:N \l_@@_final_open_bool
4259   }
4260 }
4261 \group_begin:
4262 \int_compare:nNnTF { #2 } = 0
4263 { \color { nicematrix-first-col } }

```

```

4264 {
4265   \int_compare:nNnT { #2 } = \g_@@_col_total_int
4266   { \color { nicematrix-last-col } }
4267 }
4268 \keys_set:nn { NiceMatrix / xdots } { #4 }
4269 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4270 \@@_actually_draw_Vdots:
4271 \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).

```

4272 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4273 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4274 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

4275 \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

```

4276 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4277 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

4278 \hook_gput_code:nnn { begindocument } { . }
4279 {
4280   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4281   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4282   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4283   {
4284     \group_begin:
4285     \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4286     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4287     \use:e
4288     {
4289       \@@_line_i:nn
4290       { \@@_double_int_eval:n #2 \q_stop }
4291       { \@@_double_int_eval:n #3 \q_stop }
4292     }
4293     \group_end:
4294   }
4295 }

```

⁷⁰Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

4296 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4297 {
4298   \bool_set_false:N \l_@@_initial_open_bool
4299   \bool_set_false:N \l_@@_final_open_bool
4300   \bool_if:nTF
4301     {
4302       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4303       ||
4304       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4305     }
4306     {
4307       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4308     }
4309     { \@@_draw_line_ii:nn { #1 } { #2 } }
4310   }
4311 \hook_gput_code:nnn { begindocument } { . }
4312 {
4313   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4314   {

```

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:`.

```

4315   \c_@@_pgfortikzpicture_tl
4316   \@@_draw_line_iii:nn { #1 } { #2 }
4317   \c_@@_endpgfortikzpicture_tl
4318 }
4319 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

4320 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4321 {
4322   \pgfrememberpicturepositiononpagetrue
4323   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4324   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4325   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4326   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4327   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4328   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4329   \@@_draw_line:
4330 }

```

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`

```

4331 \keys_define:nn { NiceMatrix / RowStyle }
4332 {
4333   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4334   cell-space-top-limit .initial:n = \c_zero_dim ,
4335   cell-space-top-limit .value_required:n = true ,
4336   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4337   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4338   cell-space-bottom-limit .value_required:n = true ,
4339   cell-space-limits .meta:n =
4340     {
4341       cell-space-top-limit = #1 ,
4342       cell-space-bottom-limit = #1 ,
4343     } ,
4344   color .tl_set:N = \l_tmpa_tl ,

```



```

4345 color .value_required:n = true ,
4346 bold .bool_set:N = \l_tmpa_bool ,
4347 bold .default:n = true ,
4348 bold .initial:n = false ,
4349 nb-rows .int_set:N = \l_@@_key_nb_rows_int ,
4350 nb-rows .value_required:n = true ,
4351 nb-rows .initial:n = 1 ,
4352 rowcolor .tl_set:N = \l_@@_tmpc_tl ,
4353 rowcolor .value_required:n = true ,
4354 rowcolor .initial:n = ,
4355 unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
4356 }

```

```

4357 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4358 {
4359   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

4360   \tl_if_empty:NF \l_@@_tmpc_tl
4361   {

```

First, the end of the current row (we remind that \RowStyle applies to the *end* of the current row).

```

4362     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4363     {
4364       \@@_rectanglecolor
4365       { \l_@@_tmpc_tl }
4366       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4367       { \int_use:N \c@iRow - * }
4368     }

```

Then, the other rows (if there is several rows).

```

4369     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4370     {
4371       \tl_gput_right:Nx \g_nicematrix_code_before_tl
4372       {
4373         \@@_rowcolor
4374         { \l_@@_tmpc_tl }
4375         {
4376           \int_eval:n { \c@iRow + 1 }
4377           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4378         }
4379       }
4380     }
4381   }
4382   \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4383   \tl_gput_right:Nx \g_@@_row_style_tl
4384   { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4385   \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.

```

4386   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4387   {
4388     \tl_gput_right:Nx \g_@@_row_style_tl
4389     {
4390       \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4391       {
4392         \dim_set:Nn \l_@@_cell_space_top_limit_dim
4393         { \dim_use:N \l_tmpa_dim }
4394       }
4395     }
4396   }

```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```

4397   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4398   {
4399     \tl_gput_right:Nx \g_@@_row_style_tl

```

```

4400     {
4401         \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4402         {
4403             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4404             { \dim_use:N \l_tmpb_dim }
4405         }
4406     }
4407 }
\l_tmpa_tl is the value of the key color of \RowStyle.
4408 \tl_if_empty:NF \l_tmpa_tl
4409 {
4410     \tl_gput_right:Nx \g_@@_row_style_tl
4411     { \mode_leave_vertical: \exp_not:N \color { \l_tmpa_tl } }
4412 }
\l_tmpa_bool is the value of the key bold.
4413 \bool_if:NT \l_tmpa_bool
4414 {
4415     \tl_gput_right:Nn \g_@@_row_style_tl
4416     {
4417         \if_mode_math:
4418             \c_math_toggle_token
4419             \bfseries \boldmath
4420             \c_math_toggle_token
4421         \else:
4422             \bfseries \boldmath
4423         \fi:
4424     }
4425 }
4426 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4427 \g_@@_row_style_tl
4428 \ignorespaces
4429 }

```

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).

```

4430 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4431 {

```

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.

```

4432 \int_zero:N \l_tmpa_int
4433 \seq_map_indexed_inline:Nn \g_@@_colors_seq
4434 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4435 \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

4436 {
4437   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4438   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4439 }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

4440 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4441 }
4442 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4443 \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`).

```

4444 \cs_new_protected:Npn \@@_actually_color:
4445 {
4446   \pgfpicture
4447   \pgf@relevantforpicturesizefalse
4448   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4449   {
4450     \color ##2
4451     \use:c { g_@@_color _ ##1 _tl }
4452     \tl_gclear:c { g_@@_color _ ##1 _tl }
4453     \pgfusepath { fill }
4454   }
4455   \endpgfpicture
4456 }
4457 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4458 {
4459   \tl_set:Nn \l_@@_rows_tl { #1 }
4460   \tl_set:Nn \l_@@_cols_tl { #2 }
4461   \@@_cartesian_path:
4462 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

4463 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4464 {
4465   \tl_if_blank:nF { #2 }
4466   {
4467     \@@_add_to_colors_seq:xn
4468     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4469     { \@@_cartesian_color:nn { #3 } { - } }
4470   }
4471 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

4472 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4473 {
4474   \tl_if_blank:nF { #2 }
4475   {
4476     \@@_add_to_colors_seq:xn
4477     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4478     { \@@_cartesian_color:nn { - } { #3 } }

```

```

4479     }
4480 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

4481 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4482 {
4483   \tl_if_blank:nF { #2 }
4484   {
4485     \@@_add_to_colors_seq:xn
4486     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4487     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4488   }
4489 }

```

The last argument is the radius of the corners of the rectangle.

```

4490 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4491 {
4492   \tl_if_blank:nF { #2 }
4493   {
4494     \@@_add_to_colors_seq:xn
4495     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4496     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4497   }
4498 }

```

The last argument is the radius of the corners of the rectangle.

```

4499 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4500 {
4501   \@@_cut_on_hyphen:w #1 \q_stop
4502   \tl_clear_new:N \l_@@_tmpc_tl
4503   \tl_clear_new:N \l_@@_tmpd_tl
4504   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
4505   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
4506   \@@_cut_on_hyphen:w #2 \q_stop
4507   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
4508   \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`.

```

4509   \@@_cartesian_path:n { #3 }
4510 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

4511 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4512 {
4513   \clist_map_inline:nn { #3 }
4514   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4515 }

```

```

4516 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4517 {
4518   \int_step_inline:nn { \int_use:N \c@iRow }
4519   {
4520     \int_step_inline:nn { \int_use:N \c@jCol }
4521     {
4522       \int_if_even:nTF { ####1 + ##1 }
4523       { \@@_cellcolor [ #1 ] { #2 } }
4524       { \@@_cellcolor [ #1 ] { #3 } }
4525       { ##1 - ####1 }
4526     }
4527   }
4528 }

```

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”.

```

4529 \NewDocumentCommand \@@_arraycolor { 0 { } m }
4530 {
4531   \@@_rectanglecolor [ #1 ] { #2 }
4532   { 1 - 1 }
4533   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4534 }

4535 \keys_define:nn { NiceMatrix / rowcolors }
4536 {
4537   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4538   respect-blocks .default:n = true ,
4539   cols .tl_set:N = \l_@@_cols_tl ,
4540   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4541   restart .default:n = true ,
4542   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4543 }

```

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*.

```

4544 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
4545 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

4546   \group_begin:
4547   \seq_clear_new:N \l_@@_colors_seq
4548   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
4549   \tl_clear_new:N \l_@@_cols_tl
4550   \tl_set:Nn \l_@@_cols_tl { - }
4551   \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).

```

4552   \int_zero_new:N \l_@@_color_int
4553   \int_set:Nn \l_@@_color_int 1
4554   \bool_if:NT \l_@@_respect_blocks_bool
4555   {

```

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`).

```

4556   \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4557   \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
4558   { \@@_not_in_exterior_p:nnnnn #1 }
4559 }
4560 \pgfpicture
4561 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

4562   \clist_map_inline:nn { #2 }
4563   {
4564     \tl_set:Nn \l_tmpa_tl { ##1 }
4565     \tl_if_in:NnTF \l_tmpa_tl { - }
4566     { \@@_cut_on_hyphen:w ##1 \q_stop }
4567     { \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.

```

4568 \int_set:Nn \l_tmpa_int \l_tmpa_tl
4569 \bool_if:NTF \l_@@_rowcolors_restart_bool
4570 { \int_set:Nn \l_@@_color_int 1 }
4571 { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
4572 \int_zero_new:N \l_@@_tmpc_int
4573 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
4574 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
4575 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4576 \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).

```

4577 \bool_if:NT \l_@@_respect_blocks_bool
4578 {
4579   \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
4580   { \@@_intersect_our_row_p:nnnnn ###1 }
4581   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4582 }
4583 \tl_set:Nx \l_@@_rows_tl
4584 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

4585 \tl_clear_new:N \l_@@_color_tl
4586 \tl_set:Nx \l_@@_color_tl
4587 {
4588   \@@_color_index:n
4589   {
4590     \int_mod:nn
4591     { \l_@@_color_int - 1 }
4592     { \seq_count:N \l_@@_colors_seq }
4593     + 1
4594   }
4595 }
4596 \tl_if_empty:NF \l_@@_color_tl
4597 {
4598   \@@_add_to_colors_seq:xx
4599   { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
4600   { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
4601 }
4602 \int_incr:N \l_@@_color_int
4603 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4604 }
4605 }
4606 \endpgfpicture
4607 \group_end:
4608 }

```

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.

```

4609 \cs_new:Npn \@@_color_index:n #1
4610 {
4611   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
4612   { \@@_color_index:n { #1 - 1 } }
4613   { \seq_item:Nn \l_@@_colors_seq { #1 } }
4614 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

4615 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4616 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

```

```

4617 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
4618 {
4619   \int_compare:nNt { #3 } > \l_tmpb_int
4620   { \int_set:Nn \l_tmpb_int { #3 } }
4621 }

4622 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
4623 {
4624   \bool_lazy_or:nnTF
4625   { \int_compare_p:nNn { #4 } = \c_zero_int }
4626   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
4627   \prg_return_false:
4628   \prg_return_true:
4629 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

4630 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
4631 {
4632   \bool_if:nTF
4633   {
4634     \int_compare_p:n { #1 <= \l_tmpa_int }
4635     &&
4636     \int_compare_p:n { \l_tmpa_int <= #3 }
4637   }
4638   \prg_return_true:
4639   \prg_return_false:
4640 }

```

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).

```

4641 \cs_new_protected:Npn \@@_cartesian_path:n #1
4642 {
4643   \bool_lazy_and:nnT
4644   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4645   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4646   {
4647     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4648     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4649   }

```

We begin the loop over the columns.

```

4650 \clist_map_inline:Nn \l_@@_cols_tl
4651 {
4652   \tl_set:Nn \l_tmpa_tl { ##1 }
4653   \tl_if_in:NnTF \l_tmpa_tl { - }
4654   { \@@_cut_on_hyphen:w ##1 \q_stop }
4655   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4656   \bool_lazy_or:nnT
4657   { \tl_if_blank_p:V \l_tmpa_tl }
4658   { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4659   { \tl_set:Nn \l_tmpa_tl { 1 } }
4660   \bool_lazy_or:nnT
4661   { \tl_if_blank_p:V \l_tmpb_tl }
4662   { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4663   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4664   \int_compare:nNt \l_tmpb_tl > \c@jCol
4665   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

\l_@@_tmpc_tl will contain the number of column.

```
4666 \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.

```
4667 \@@_qpoint:n { col - \l_tmpa_tl }
4668 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4669 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4670 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4671 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
4672 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
4673 \clist_map_inline:Nn \l_@@_rows_tl
4674 {
4675   \tl_set:Nn \l_tmpa_tl { #####1 }
4676   \tl_if_in:NnTF \l_tmpa_tl { - }
4677   { \@@_cut_on_hyphen:w #####1 \q_stop }
4678   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4679   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4680   \tl_if_empty:NT \l_tmpb_tl
4681   { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4682   \int_compare:nNnT \l_tmpb_tl > \c@iRow
4683   { \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.

```
4684 \seq_if_in:NxF \l_@@_corners_cells_seq
4685 { \l_tmpa_tl - \l_@@_tmpc_tl }
4686 {
4687   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
4688   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4689   \@@_qpoint:n { row - \l_tmpa_tl }
4690   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4691   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4692   \pgfpathrectanglecorners
4693   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
4694   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4695 }
4696 }
4697 }
4698 }
```

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).

```
4699 \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.

```
4700 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4701 {
4702   \clist_set_eq:NN \l_tmpa_clist #1
4703   \clist_clear:N #1
4704   \clist_map_inline:Nn \l_tmpa_clist
4705   {
4706     \tl_set:Nn \l_tmpa_tl { ##1 }
4707     \tl_if_in:NnTF \l_tmpa_tl { - }
4708     { \@@_cut_on_hyphen:w ##1 \q_stop }
4709     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4710     \bool_lazy_or:nnT
4711     { \tl_if_blank_p:V \l_tmpa_tl }
4712     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
```



```

4713     { \tl_set:Nn \l_tmpa_tl { 1 } }
4714   \bool_lazy_or:nnT
4715     { \tl_if_blank_p:V \l_tmpb_tl }
4716     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4717     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4718   \int_compare:nNnT \l_tmpb_tl > #2
4719     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4720   \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4721     { \clist_put_right:Nn #1 { ####1 } }
4722 }
4723 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the `tabular`.

```

4724 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4725 {
4726   \peek_remove_spaces:n
4727   {
4728     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4729     {

```

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`).

```

4730       \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
4731       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4732     }
4733   }
4734 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\rowcolor` in the `tabular`.

```

4735 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4736 {
4737   \peek_remove_spaces:n
4738   {
4739     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4740     {
4741       \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4742       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4743       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4744     }
4745   }
4746 }

```

```

4747 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4748 {

```

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).

```

4749   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4750   {

```

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).

```

4751     \tl_gput_left:Nx \g_nicematrix_code_before_tl
4752     {
4753       \exp_not:N \columncolor [ #1 ]
4754       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4755     }
4756   }
4757 }

```

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`.

```
4758 \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`.

```
4759 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4760 {
4761   \int_compare:nNnTF \l_@@_first_col_int = 0
4762     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4763     {
4764       \int_compare:nNnTF \c@jCol = 0
4765         {
4766           \int_compare:nNnF \c@iRow = { -1 }
4767             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4768         }
4769       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4770     }
4771 }
```

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.

```
4772 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4773 {
4774   \int_compare:nNnF \c@iRow = 0
4775     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4776 }
```

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.

```
4777 \keys_define:nn { NiceMatrix / Rules }
4778 {
4779   position .int_set:N = \l_@@_position_int ,
4780   position .value_required:n = true ,
4781   start .int_set:N = \l_@@_start_int ,
4782   start .initial:n = 1 ,
4783   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`).

```
4784   % letter .code:n = \prg_do_nothing: ,
4785   % command .code:n = \prg_do_nothing:
4786 }
```

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.

```

4787 \keys_define:nn { NiceMatrix / RulesBis }
4788 {
4789   multiplicity .int_set:N = \l_@@_multiplicity_int ,
4790   multiplicity .initial:n = 1 ,
4791   dotted .bool_set:N = \l_@@_dotted_bool ,
4792   dotted .initial:n = false ,
4793   dotted .default:n = true ,
4794   color .code:n = \@@_set_CT@arc@: #1 \q_stop ,
4795   color .value_required:n = true ,
4796   sep-color .code:n = \@@_set_CT@drsc@: #1 \q_stop ,
4797   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.

```

4798   tikz .tl_set:N = \l_@@_tikz_rule_tl ,
4799   tikz .value_required:n = true ,
4800   tikz .initial:n = ,
4801   width .dim_set:N = \l_@@_rule_width_dim ,
4802   width .value_required:n = true
4803 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

4804 \cs_new_protected:Npn \@@_vline:n #1
4805 {

```

The group is for the options.

```

4806   \group_begin:
4807   \int_zero_new:N \l_@@_end_int
4808   \int_set_eq:NN \l_@@_end_int \c@iRow
4809   \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).

```

4810   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
4811   \@@_vline_i:
4812   \group_end:
4813 }

```

```

4814 \cs_new_protected:Npn \@@_vline_i:
4815 {
4816   \int_zero_new:N \l_@@_local_start_int
4817   \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`.

```

4818   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
4819   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
4820     \l_tmpa_tl
4821   {

```

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.

```

4822     \bool_gset_true:N \g_tmpa_bool
4823     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq

```

```

4824     { \@@_test_vline_in_block:nnnnn ##1 }
4825 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4826     { \@@_test_vline_in_block:nnnnn ##1 }
4827 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4828     { \@@_test_vline_in_stroken_block:nnnn ##1 }
4829 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
4830 \bool_if:NTF \g_tmpa_bool
4831     {
4832         \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.

```

4833         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
4834     }
4835     {
4836         \int_compare:nNnT \l_@@_local_start_int > 0
4837         {
4838             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
4839             \@@_vline_ii:
4840             \int_zero:N \l_@@_local_start_int
4841         }
4842     }
4843 }
4844 \int_compare:nNnT \l_@@_local_start_int > 0
4845 {
4846     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
4847     \@@_vline_ii:
4848 }
4849 }

4850 \cs_new_protected:Npn \@@_test_in_corner_v:
4851 {
4852     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
4853     {
4854         \seq_if_in:NxT
4855         \l_@@_corners_cells_seq
4856         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
4857         { \bool_set_false:N \g_tmpa_bool }
4858     }
4859     {
4860         \seq_if_in:NxT
4861         \l_@@_corners_cells_seq
4862         { \l_tmpa_tl - \l_tmpb_tl }
4863         {
4864             \int_compare:nNnTF \l_tmpb_tl = 1
4865             { \bool_set_false:N \g_tmpa_bool }
4866             {
4867                 \seq_if_in:NxT
4868                 \l_@@_corners_cells_seq
4869                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
4870                 { \bool_set_false:N \g_tmpa_bool }
4871             }
4872         }
4873     }
4874 }

4875 \cs_new_protected:Npn \@@_vline_ii:
4876 {
4877     \bool_set_false:N \l_@@_dotted_bool

```

We use \keys_set_known:nV and not \keys_set:nV because there may be the keys **letter** and **command** in the list (these keys are present if the rule comes from a customized line (created by custom-line)).

```

4878 \keys_set_known:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
4879 \bool_if:NTF \l_@@_dotted_bool
4880 \@@_vline_iv:
4881 {
4882   \tl_if_empty:NTF \l_@@_tikz_rule_tl
4883   \@@_vline_iii:
4884   \@@_vline_v:
4885 }
4886 }

```

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`).

```

4887 \cs_new_protected:Npn \@@_vline_iii:
4888 {
4889   \pgfpicture
4890   \pgfrememberpicturepositiononpagetrue
4891   \pgf@relevantforpicturesizefalse
4892   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4893   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4894   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4895   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4896   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4897   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4898   \bool_lazy_all:nT
4899   {
4900     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
4901     { \cs_if_exist_p:N \CT@drsc@ }
4902     { ! \tl_if_blank_p:V \CT@drsc@ }
4903   }
4904   {
4905     \group_begin:
4906     \CT@drsc@
4907     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4908     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
4909     \dim_set:Nn \l_@@_tmpd_dim
4910     {
4911       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
4912       * ( \l_@@_multiplicity_int - 1 )
4913     }
4914     \pgfpathrectanglecorners
4915     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4916     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
4917     \pgfusepath { fill }
4918     \group_end:
4919   }
4920   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4921   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
4922   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
4923   {
4924     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4925     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4926     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4927     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
4928   }
4929   \CT@arc@
4930   \pgfsetlinewidth { 1.1 \arrayrulewidth }
4931   \pgfsetrectcap
4932   \pgfusepathqstroke
4933   \endpgfpicture
4934 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

4935 \cs_new_protected:Npn \l_@@_vline_iv:
4936 {
4937   \pgfpicture
4938   \pgfrememberpicturepositiononpagetrue
4939   \pgf@relevantforpicturesizefalse
4940   \l_@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4941   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4942   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4943   \l_@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4944   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4945   \l_@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4946   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4947   \CT@arc@
4948   \l_@@_draw_line:
4949   \endpgfpicture
4950 }

```

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`).

```

4951 \cs_new_protected:Npn \l_@@_vline_v:
4952 {
4953   \begin {tikzpicture }
4954   \pgfrememberpicturepositiononpagetrue
4955   \pgf@relevantforpicturesizefalse
4956   \l_@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4957   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4958   \l_@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4959   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
4960   \l_@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4961   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4962   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
4963   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
4964     ( \l_tmpb_dim , \l_tmpa_dim ) --
4965     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
4966   \end { tikzpicture }
4967 }

```

The command `\l_@@_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).

```

4968 \cs_new_protected:Npn \l_@@_draw_vlines:
4969 {
4970   \int_step_inline:nnn
4971     {
4972       \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4973       1 2
4974     }
4975     {
4976       \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4977       { \int_eval:n { \c@jCol + 1 } }
4978       \c@jCol
4979     }
4980     {
4981       \tl_if_eq:NnF \l_@@_vlines_clist { all }
4982       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4983       { \l_@@_vline:n { position = ##1 } }
4984     }
4985 }

```

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}`.

```

4986 \cs_new_protected:Npn \@@_hline:n #1
4987 {

```

The group is for the options.

```

4988 \group_begin:
4989 \int_zero_new:N \l_@@_end_int
4990 \int_set_eq:NN \l_@@_end_int \c@jCol
4991 \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
4992 \@@_hline_i:
4993 \group_end:
4994 }

4995 \cs_new_protected:Npn \@@_hline_i:
4996 {
4997 \int_zero_new:N \l_@@_local_start_int
4998 \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 .

```

4999 \tl_set:Nx \l\_tmpa\_tl { \int_use:N \l_@@_position_int }
5000 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5001 \l\_tmpb\_tl
5002 {

```

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.

```

5003 \bool_gset_true:N \g\_tmpa\_bool
5004 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5005 { \@@_test_hline_in_block:nnnnn ##1 }
5006 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5007 { \@@_test_hline_in_block:nnnnn ##1 }
5008 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5009 { \@@_test_hline_in_stroken_block:nnnn ##1 }
5010 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5011 \bool_if:NTF \g\_tmpa\_bool
5012 {
5013 \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. \l_tmpc_tl will be the starting row of the rule that we will have to draw.

```

5014 { \int_set:Nn \l_@@_local_start_int \l\_tmpc\_tl }
5015 }
5016 {
5017 \int_compare:nNnT \l_@@_local_start_int > 0
5018 {
5019 \int_set:Nn \l_@@_local_end_int { \l\_tmpc\_tl - 1 }
5020 \@@_hline_ii:
5021 \int_zero:N \l_@@_local_start_int
5022 }
5023 }
5024 }
5025 \int_compare:nNnT \l_@@_local_start_int > 0
5026 {
5027 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5028 \@@_hline_ii:
5029 }
5030 }

```

```

5031 \cs_new_protected:Npn \@@_test_in_corner_h:
5032 {
5033 \int_compare:nNnTF \l\_tmpa\_tl = { \int_eval:n { \c@iRow + 1 } }
5034 {

```

```

5035     \seq_if_in:NxT
5036     \l_@@_corners_cells_seq
5037     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5038     { \bool_set_false:N \g_tmpa_bool }
5039   }
5040   {
5041     \seq_if_in:NxT
5042     \l_@@_corners_cells_seq
5043     { \l_tmpa_tl - \l_tmpb_tl }
5044     {
5045       \int_compare:nNnTF \l_tmpa_tl = 1
5046       { \bool_set_false:N \g_tmpa_bool }
5047       {
5048         \seq_if_in:NxT
5049         \l_@@_corners_cells_seq
5050         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5051         { \bool_set_false:N \g_tmpa_bool }
5052       }
5053     }
5054   }
5055 }

```

```

5056 \cs_new_protected:Npn \@@_hline_ii:
5057 {
5058   \bool_set_false:N \l_@@_dotted_bool

```

We use `\keys_set_known:nV` and not `\keys_set:nV` because there may be the keys `letter` and `command` in the list (these keys are present if the rule comes from a customized line (created by `custom-line`)).

```

5059   \keys_set_known:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5060   \bool_if:NTF \l_@@_dotted_bool
5061   \@@_hline_iv:
5062   {
5063     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5064     \@@_hline_iii:
5065     \@@_hline_v:
5066   }
5067 }

```

First the case of a standard rule, that is to say a rule which is not dotted.

```

5068 \cs_new_protected:Npn \@@_hline_iii:
5069 {
5070   \pgfpicture
5071   \pgfrememberpicturepositiononpagetrue
5072   \pgf@relevantforpicturesizefalse
5073   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5074   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5075   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5076   \dim_set_eq:NN \l_tmpb_dim \pgf@y
5077   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5078   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5079   \bool_lazy_all:nT
5080   {
5081     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5082     { \cs_if_exist_p:N \CT@drsc@ }
5083     { ! \tl_if_blank_p:V \CT@drsc@ }
5084   }
5085   {
5086     \group_begin:
5087     \CT@drsc@
5088     \dim_set:Nn \l_@@_tmpd_dim
5089     {

```



```

5090         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5091         * ( \l_@@_multiplicity_int - 1 )
5092     }
5093     \pgfpathrectanglecorners
5094     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5095     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5096     \pgfusepathqfill
5097     \group_end:
5098 }
5099 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5100 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5101 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5102 {
5103     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5104     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5105     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5106     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5107 }
5108 \CT@arc@
5109 \pgfsetlinewidth { 1.1 \arrayrulewidth }
5110 \pgfsetrectcap
5111 \pgfusepathqstroke
5112 \endpgfpicture
5113 }

```

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}
```

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

```
5114 \cs_new_protected:Npn \l_@@_hline_iv:
```

```
5115 {
```

```
5116     \pgfpicture
```

```
5117     \pgfrememberpicturepositiononpagetrue
```

```
5118     \pgf@relevantforpicturesizefalse
```

```
5119     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
```

```
5120     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
```

```
5121     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
```

```
5122     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
```

```
5123     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
```

```
5124     \int_compare:nNt \l_@@_local_start_int = 1
```

```
5125     {
```

```
5126         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
```

```
5127         \bool_if:NT \l_@@_NiceArray_bool
```

```
5128         { \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.

```
5129     \tl_if_eq:NnF \g_@@_left_delim_tl (
```

```
5130         { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
```

```

5131     }
5132     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5133     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5134     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5135     {
5136         \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5137         \bool_if:NT \l_@@_NiceArray_bool
5138         { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5139         \tl_if_eq:NnF \g_@@_right_delim_tl )
5140         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
5141     }
5142     \CT@arc@
5143     \@@_draw_line:
5144     \endpgfpicture
5145 }

```

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`).

```

5146 \cs_new_protected:Npn \@@_hline_v:
5147 {
5148     \begin { tikzpicture }
5149     \pgfrememberpicturepositiononpagetrue
5150     \pgf@relevantforpicturesizefalse
5151     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5152     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5153     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5154     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
5155     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5156     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5157     \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5158     \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5159     ( \l_tmpa_dim , \l_tmpb_dim ) --
5160     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5161     \end { tikzpicture }
5162 }

```

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)).

```

5163 \cs_new_protected:Npn \@@_draw_hlines:
5164 {
5165     \int_step_inline:nnn
5166     {
5167         \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5168         1 2
5169     }
5170     {
5171         \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5172         { \int_eval:n { \c@iRow + 1 } }
5173         \c@iRow
5174     }
5175     {
5176         \tl_if_eq:NnF \l_@@_hlines_clist { all }
5177         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5178         { \@@_hline:n { position = ##1 } }
5179     }
5180 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5181 \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.

```

5182 \cs_set:Npn \@@_Hline_i:n #1
5183 {
5184   \peek_meaning_ignore_spaces:NTF \Hline
5185   { \@@_Hline_ii:nn { #1 + 1 } }
5186   { \@@_Hline_iii:n { #1 } }
5187 }
5188 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5189 \cs_set:Npn \@@_Hline_iii:n #1
5190 {
5191   \skip_vertical:n
5192   {
5193     \arrayrulewidth * ( #1 )
5194     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
5195   }
5196   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5197   {
5198     \@@_hline:n
5199     {
5200       position = \int_eval:n { \c@iRow + 1 } ,
5201       multiplicity = #1
5202     }
5203   }
5204   \ifnum 0 = `{ \fi }
5205 }

```

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).

```

5206 \keys_define:nn { NiceMatrix / ColumnTypes } { }

```

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```

5207 \cs_new_protected:Npn \@@_custom_line:n #1
5208 {
5209   \str_clear_new:N \l_@@_command_str
5210   \str_clear_new:N \l_@@_letter_str
5211   \dim_zero_new:N \l_@@_rule_width_dim

```

The token list `\l_tmpa_tl` is for the key `color`.

```

5212   \tl_clear:N \l_tmpa_tl

```

The flag `\l_tmpa_bool` will indicate whether the key `tikz` is present.

```

5213   \bool_set_false:N \l_tmpa_bool

```

The flag `\l_tmpb_bool` will indicate whether the key `width` is present.

```

5214   \bool_set_false:N \l_tmpb_bool
5215   \keys_set_known:nn { NiceMatrix / Custom-Line } { #1 }
5216   \bool_if:NT \l_tmpa_bool
5217   {

```

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.

```

5218     \cs_if_exist:NF \tikzpicture
5219     { \@@_error:n { tikz-in-custom-line-without-tikz } }
5220     \tl_if_empty:NF \l_tmpa_tl

```

```

5221         { \@@_error:n { color~in~custom~line~with~tikz } }
5222     }
5223     \bool_if:NT \l_tmpb_bool
5224     {
5225         \bool_if:NF \l_tmpa_bool
5226         { \@@_error:n { key~width~without~key~tikz } }
5227     }

```

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.

```

5228     \bool_lazy_and:nnTF
5229     { \str_if_empty_p:N \l_@@_letter_str }
5230     { \str_if_empty_p:N \l_@@_command_str }
5231     { \@@_error:n { No~letter~and~no~command } }
5232     {
5233         \str_if_empty:NF \l_@@_letter_str
5234         {
5235             \int_compare:nNnTF { \str_count:N \l_@@_letter_str } = 1
5236             {
5237                 \exp_args:NnV \tl_if_in:NnTF
5238                 \c_@@_forbidden_letters_str \l_@@_letter_str
5239                 { \@@_error:n { Forbidden~letter } }
5240             }

```

The final user can, locally, redefine a letter of column type. That's compatible with the use of `\keys_define:nn`: the definition is local and may overwrite a previous definition.

```

5241         \keys_define:nx { NiceMatrix / ColumnTypes }
5242         {
5243             \l_@@_letter_str .code:n =
5244             { \@@_custom_line_i:n { \exp_not:n { #1 } } }
5245         }
5246     }
5247 }
5248 { \@@_error:n { Several~letters } }
5249 }
5250 \str_if_empty:NF \l_@@_command_str
5251 {

```

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.

```

5252     \bool_if:NF \l_tmpa_bool
5253     {
5254         \dim_set:Nn \l_@@_rule_width_dim
5255         {
5256             \arrayrulewidth * \l_@@_tmpc_int
5257             + \doublerulesep * ( \l_@@_tmpc_int - 1 )
5258         }
5259     }
5260     \@@_define_h_custom_line:nV { #1 } \l_@@_rule_width_dim
5261 }
5262 }
5263 }
5264 \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.

```

5265 \keys_define:nn { NiceMatrix / Custom-Line }
5266 {
5267     % here, we will use change in the future to use .tl_set:N

```

```

5268 letter .code:n = \str_set:Nn \l_@@_letter_str { #1 } ,
5269 letter .value_required:n = true ,
5270 % here, we will use change in the future to use .tl_set:N
5271 command .code:n = \str_set:Nn \l_@@_command_str { #1 } ,
5272 command .value_required:n = true ,
5273 multiplicity .int_set:N = \l_@@_tmpc_int ,
5274 multiplicity .initial:n = 1 ,
5275 multiplicity .value_required:n = true ,
5276 color .tl_set:N = \l_tmpa_tl ,
5277 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`.

```

5278 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).

```

5279 width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5280 \bool_set_true:N \l_tmpb_bool ,
5281 width .value_required:n = true ,
5282 unknown .code:n = \@@_error:n { Unknown-key-for-custom-line }
5283 }

```

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.

```

5284 \cs_new_protected:Npn \@@_define_h_custom_line:nn #1 #2
5285 {

```

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`.

```

5286 \cs_set:cpn { nicematrix - \l_@@_command_str }
5287 {
5288 \noalign
5289 {
5290 \skip_vertical:n { #2 }
5291 \tl_gput_right:Nx \g_@@_internal_code_after_tl
5292 { \@@_hline:n { #1 , position = \int_eval:n { \c@iRow + 1 } } }
5293 }
5294 }
5295 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
5296 }
5297 \cs_generate_variant:Nn \@@_define_h_custom_line:nn { n V }

```

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.

```

5298 \cs_new_protected:Npn \@@_custom_line_i:n #1
5299 {
5300 \bool_if:NF \l_tmpa_bool
5301 {
5302 \dim_set:Nn \l_@@_rule_width_dim
5303 {
5304 \arrayrulewidth * \l_@@_tmpc_int
5305 + \doublerulesep * ( \l_@@_tmpc_int - 1 )
5306 }
5307 }
5308 \tl_gput_right:Nx \g_@@_preamble_tl
5309 {
5310 \exp_not:N !
5311 { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } }
5312 }
5313 \tl_gput_right:Nx \g_@@_internal_code_after_tl
5314 { \@@_vline:n { #1 , position = \int_eval:n { \c@jCol + 1 } } }
5315 }

```

```
5316 \@@_custom_line:n { letter = : , command = hdottedline , dotted }
```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to false.

```
5317 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4
5318 {
5319   \bool_lazy_all:nT
5320   {
5321     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
5322     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5323     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5324     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5325   }
5326   { \bool_gset_false:N \g_tmpa_bool }
5327 }
```

The same for vertical rules.

```
5328 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4
5329 {
5330   \bool_lazy_all:nT
5331   {
5332     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5333     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5334     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
5335     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5336   }
5337   { \bool_gset_false:N \g_tmpa_bool }
5338 }
5339 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
5340 {
5341   \bool_lazy_all:nT
5342   {
5343     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5344     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
5345     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5346     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5347   }
5348   { \bool_gset_false:N \g_tmpa_bool }
5349 }
5350 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
5351 {
5352   \bool_lazy_all:nT
5353   {
5354     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5355     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5356     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5357     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
5358   }
5359   { \bool_gset_false:N \g_tmpa_bool }
5360 }
```

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.

```
5361 \cs_new_protected:Npn \@@_compute_corners:
5362 {
```

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.

```

5363 \seq_clear_new:N \l_@@_corners_cells_seq
5364 \clist_map_inline:Nn \l_@@_corners_clist
5365 {
5366   \str_case:nnF { ##1 }
5367   {
5368     { NW }
5369     { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
5370     { NE }
5371     { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
5372     { SW }
5373     { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
5374     { SE }
5375     { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
5376   }
5377   { \@@_error:nn { bad~corner } { ##1 } }
5378 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

5379 \seq_if_empty:NF \l_@@_corners_cells_seq
5380 {

```

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.

```

5381 \tl_gput_right:Nx \g_@@_aux_tl
5382 {
5383   \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
5384   { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
5385 }
5386 }
5387 }

```

“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.

```

5388 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
5389 {

```

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.

```

5390 \bool_set_false:N \l_tmpa_bool
5391 \int_zero_new:N \l_@@_last_empty_row_int
5392 \int_set:Nn \l_@@_last_empty_row_int { #1 }
5393 \int_step_inline:nnnn { #1 } { #3 } { #5 }
5394 {
5395   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
5396   \bool_lazy_or:nnTF
5397   {
5398     \cs_if_exist_p:c
5399     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }

```

```

5400     }
5401     \l_tmpb_bool
5402     { \bool_set_true:N \l_tmpa_bool }
5403     {
5404         \bool_if:NF \l_tmpa_bool
5405         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
5406     }
5407 }

```

Now, you determine the last empty cell in the row of number 1.

```

5408     \bool_set_false:N \l_tmpa_bool
5409     \int_zero_new:N \l_@@_last_empty_column_int
5410     \int_set:Nn \l_@@_last_empty_column_int { #2 }
5411     \int_step_inline:nnnn { #2 } { #4 } { #6 }
5412     {
5413         \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
5414         \bool_lazy_or:nnTF
5415         \l_tmpb_bool
5416         {
5417             \cs_if_exist_p:c
5418             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
5419         }
5420         { \bool_set_true:N \l_tmpa_bool }
5421         {
5422             \bool_if:NF \l_tmpa_bool
5423             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
5424         }
5425     }

```

Now, we loop over the rows.

```

5426     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
5427     {

```

We treat the row number ##1 with another loop.

```

5428         \bool_set_false:N \l_tmpa_bool
5429         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
5430         {
5431             \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
5432             \bool_lazy_or:nnTF
5433             \l_tmpb_bool
5434             {
5435                 \cs_if_exist_p:c
5436                 { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
5437             }
5438             { \bool_set_true:N \l_tmpa_bool }
5439             {
5440                 \bool_if:NF \l_tmpa_bool
5441                 {
5442                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
5443                     \seq_put_right:Nn
5444                     \l_@@_corners_cells_seq
5445                     { ##1 - #####1 }
5446                 }
5447             }
5448         }
5449     }
5450 }

```

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).

```

5451 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
5452 {
5453     \int_set:Nn \l_tmpa_int { #1 }

```



```

5454 \int_set:Nn \l_tmpb_int { #2 }
5455 \bool_set_false:N \l_tmpb_bool
5456 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5457 { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
5458 }
5459 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
5460 {
5461   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
5462   {
5463     \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
5464     {
5465       \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
5466       {
5467         \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
5468         { \bool_set_true:N \l_tmpb_bool }
5469       }
5470     }
5471   }
5472 }

```

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`.

```

5473 \cs_new:Npn \@@_hdottedline:
5474 {
5475   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
5476   \@@_hdottedline_i:
5477 }

```

On the other side, the following command should be protected.

```

5478 \cs_new_protected:Npn \@@_hdottedline_i:
5479 {

```

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).

```

5480 \tl_gput_right:Nx \g_@@_internal_code_after_tl
5481 { \@@_hdottedline:n { \int_use:N \c@iRow } }
5482 }

```

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.

```

5483 \cs_new_protected:Npn \@@_hdottedline:n #1
5484 { \@@_hline:n { position = #1 , end = \int_use:N \c@jCol , dotted } }

```

Vertical dotted lines

```

5485 \cs_new_protected:Npn \@@_vdottedline:n #1
5486 { \@@_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.

```
5487 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
5488 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5489 {
5490   auto-columns-width .code:n =
5491   {
5492     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5493     \dim_gzero_new:N \g_@@_max_cell_width_dim
5494     \bool_set_true:N \l_@@_auto_columns_width_bool
5495   }
5496 }

5497 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
5498 {
5499   \int_gincr:N \g_@@_NiceMatrixBlock_int
5500   \dim_zero:N \l_@@_columns_width_dim
5501   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5502   \bool_if:NT \l_@@_block_auto_columns_width_bool
5503   {
5504     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5505     {
5506       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
5507       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5508     }
5509   }
5510 }
```

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).

```
5511 {
5512   \bool_if:NT \l_@@_block_auto_columns_width_bool
5513   {
5514     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5515     \iow_shipout:Nx \@mainaux
5516     {
5517       \cs_gset:cpn
5518       { @@_max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```
5519       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5520     }
5521     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5522   }
5523 }
```

The extra nodes

First, two variants of the functions \dim_min:nn and \dim_max:nn.

```
5524 \cs_generate_variant:Nn \dim_min:nn { v n }
5525 \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).

```

5526 \cs_new_protected:Npn \@@_create_extra_nodes:
5527 {
5528   \bool_if:NTF \l_@@_medium_nodes_bool
5529   {
5530     \bool_if:NTF \l_@@_large_nodes_bool
5531     \@@_create_medium_and_large_nodes:
5532     \@@_create_medium_nodes:
5533   }
5534   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5535 }

```

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`.

```

5536 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5537 {
5538   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5539   {
5540     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
5541     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
5542     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
5543     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
5544   }
5545   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5546   {
5547     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
5548     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
5549     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
5550     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
5551   }

```

We begin the two nested loops over the rows and the columns of the array.

```

5552   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5553   {
5554     \int_step_variable:nnNn
5555     \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.

```

5556     {
5557       \cs_if_exist:cT
5558       { 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`.

```

5559     {
5560       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5561       \dim_set:cn { l_@@_row\_@@_i: _min_dim }

```

```

5562         { \dim_min:vn { l_@@_row _ @@_i: _min_dim } \pgf@y }
5563     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { @@_i: - @@_j: }
5564     {
5565         \dim_set:cn { l_@@_column _ @@_j: _min_dim }
5566         { \dim_min:vn { l_@@_column _ @@_j: _min_dim } \pgf@x }
5567     }

```

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`.

```

5568     \pgfpointanchor { @@_env: - @@_i: - @@_j: } { north-east }
5569     \dim_set:cn { l_@@_row _ @@_i: _ max_dim }
5570     { \dim_max:vn { l_@@_row _ @@_i: _ max_dim } \pgf@y }
5571     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { @@_i: - @@_j: }
5572     {
5573         \dim_set:cn { l_@@_column _ @@_j: _ max_dim }
5574         { \dim_max:vn { l_@@_column _ @@_j: _ max_dim } \pgf@x }
5575     }
5576 }
5577 }
5578 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

5579     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int @@_i:
5580     {
5581         \dim_compare:nNnT
5582         { \dim_use:c { l_@@_row _ @@_i: _ min _ dim } } = \c_max_dim
5583         {
5584             @@_qpoint:n { row - @@_i: - base }
5585             \dim_set:cn { l_@@_row _ @@_i: _ max _ dim } \pgf@y
5586             \dim_set:cn { l_@@_row _ @@_i: _ min _ dim } \pgf@y
5587         }
5588     }
5589     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int @@_j:
5590     {
5591         \dim_compare:nNnT
5592         { \dim_use:c { l_@@_column _ @@_j: _ min _ dim } } = \c_max_dim
5593         {
5594             @@_qpoint:n { col - @@_j: }
5595             \dim_set:cn { l_@@_column _ @@_j: _ max _ dim } \pgf@y
5596             \dim_set:cn { l_@@_column _ @@_j: _ min _ dim } \pgf@y
5597         }
5598     }
5599 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

5600 \cs_new_protected:Npn \@@_create_medium_nodes:
5601 {
5602     \pgfpicture
5603     \pgfrememberpicturepositiononpagetrue
5604     \pgf@relevantforpicturesizefalse
5605     \@@_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”.

```

5606     \tl_set:Nn \l_@@_suffix_tl { -medium }
5607     \@@_create_nodes:
5608     \endpgfpicture
5609 }

```

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:`.

```

5610 \cs_new_protected:Npn \@@_create_large_nodes:
5611 {
5612   \pgfpicture
5613     \pgfrememberpicturepositiononpagetrue
5614     \pgf@relevantforpicturesizefalse
5615     \@@_computations_for_medium_nodes:
5616     \@@_computations_for_large_nodes:
5617     \tl_set:Nn \l_@@_suffix_tl { - large }
5618     \@@_create_nodes:
5619   \endpgfpicture
5620 }

5621 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5622 {
5623   \pgfpicture
5624     \pgfrememberpicturepositiononpagetrue
5625     \pgf@relevantforpicturesizefalse
5626     \@@_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”.

```

5627   \tl_set:Nn \l_@@_suffix_tl { - medium }
5628   \@@_create_nodes:
5629   \@@_computations_for_large_nodes:
5630   \tl_set:Nn \l_@@_suffix_tl { - large }
5631   \@@_create_nodes:
5632 \endpgfpicture
5633 }

```

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.

```

5634 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5635 {
5636   \int_set:Nn \l_@@_first_row_int 1
5637   \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`.

```

5638   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5639   {
5640     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5641     {
5642       (
5643         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5644         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
5645       )
5646       / 2
5647     }
5648     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
5649     { l_@@_row _ \@@_i: _ min _ dim }
5650   }
5651   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5652   {
5653     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5654     {

```

⁷¹If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

5655      (
5656      \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5657      \dim_use:c
5658      { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
5659      )
5660      / 2
5661      }
5662      \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
5663      { l_@@_column _ \@@_j: _ max _ dim }
5664      }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

5665      \dim_sub:cn
5666      { l_@@_column _ 1 _ min _ dim }
5667      \l_@@_left_margin_dim
5668      \dim_add:cn
5669      { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5670      \l_@@_right_margin_dim
5671      }

```

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).

```

5672 \cs_new_protected:Npn \@@_create_nodes:
5673 {
5674   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5675   {
5676     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5677     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

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 _ \@@_j: _ 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         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5689       }
5690     }
5691   }

```

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 .

```

5692   \seq_mapthread_function:NNN
5693   \g_@@_multicolumn_cells_seq
5694   \g_@@_multicolumn_sizes_seq
5695   \@@_node_for_multicolumn:nn
5696 }

```

```

5697 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5698 {
5699   \cs_set_nopar:Npn \@@_i: { #1 }
5700   \cs_set_nopar:Npn \@@_j: { #2 }
5701 }

```

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”).

```

5702 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5703 {
5704   \@@_extract_coords_values: #1 \q_stop
5705   \@@_pgf_rect_node:nnnnn
5706     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5707     { \dim_use:c { \l_@@_column _ \@@_j: _ min _ dim } }
5708     { \dim_use:c { \l_@@_row _ \@@_i: _ min _ dim } }
5709     { \dim_use:c { \l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
5710     { \dim_use:c { \l_@@_row _ \@@_i: _ max _ dim } }
5711   \str_if_empty:NF \l_@@_name_str
5712   {
5713     \pgfnodealias
5714       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5715       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
5716   }
5717 }

```

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.

```

5718 \keys_define:nn { NiceMatrix / Block / FirstPass }
5719 {
5720   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5721   l .value_forbidden:n = true ,
5722   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5723   r .value_forbidden:n = true ,
5724   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5725   c .value_forbidden:n = true ,
5726   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5727   L .value_forbidden:n = true ,
5728   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5729   R .value_forbidden:n = true ,
5730   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5731   C .value_forbidden:n = true ,
5732   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5733   t .value_forbidden:n = true ,
5734   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5735   b .value_forbidden:n = true ,
5736   color .tl_set:N = \l_@@_color_tl ,
5737   color .value_required:n = true ,
5738   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
5739   respect-arraystretch .default:n = true ,
5740 }

```

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.

```

5741 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
5742 {

```

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).

```

5743   \peek_remove_spaces:n

```

```

5744 {
5745   \tl_if_blank:nTF { #2 }
5746     { \@@_Block_i 1-1 \q_stop }
5747     { \@@_Block_i #2 \q_stop }
5748     { #1 } { #3 } { #4 }
5749   }
5750 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5751 \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.

```

5752 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5753 {

```

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).

```

5754   \bool_lazy_or:nnTF
5755     { \tl_if_blank_p:n { #1 } }
5756     { \str_if_eq_p:nn { #1 } { * } }
5757     { \int_set:Nn \l_tmpa_int { 100 } }
5758     { \int_set:Nn \l_tmpa_int { #1 } }
5759   \bool_lazy_or:nnTF
5760     { \tl_if_blank_p:n { #2 } }
5761     { \str_if_eq_p:nn { #2 } { * } }
5762     { \int_set:Nn \l_tmpb_int { 100 } }
5763     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5764   \int_compare:nNnTF \l_tmpb_int = 1
5765   {
5766     \str_if_empty:NTF \l_@@_hpos_cell_str
5767       { \str_set:Nn \l_@@_hpos_block_str c }
5768       { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
5769   }
5770   { \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.

```

5771   \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
5772   \tl_set:Nx \l_tmpa_tl
5773   {
5774     { \int_use:N \c@iRow }
5775     { \int_use:N \c@jCol }
5776     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5777     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5778   }

```

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).

```

5779   \bool_if:nTF

```



```

5780 {
5781   (
5782     \int_compare_p:nNn { \l_tmpa_int } = 1
5783     ||
5784     \int_compare_p:nNn { \l_tmpb_int } = 1
5785   )
5786   && ! \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.

```

5787     && ! \l_@@_X_column_bool
5788   }
5789   { \exp_args:Nxx \@@_Block_iv:nnnnn }
5790   { \exp_args:Nxx \@@_Block_v:nnnnn }
5791   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5792 }

```

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).

```

5793 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5794 {
5795   \int_gincr:N \g_@@_block_box_int
5796   \set_protected_nopar:Npn \diagbox ##1 ##2
5797   {
5798     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5799     {
5800       \@@_actually_diagbox:nnnnnn
5801       { \int_use:N \c@iRow }
5802       { \int_use:N \c@jCol }
5803       { \int_eval:n { \c@iRow + #1 - 1 } }
5804       { \int_eval:n { \c@jCol + #2 - 1 } }
5805       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5806     }
5807   }
5808   \box_gclear_new:c
5809   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
5810   \hbox_gset:cn
5811   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
5812   {

```

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}).

```

5813     \tl_if_empty:NTF \l_@@_color_tl
5814     { \int_compare:nNnT { #2 } = 1 \set@color }
5815     { \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.

```

5816     \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5817     \group_begin:
5818     \bool_if:NF \l_@@_respect_arraystretch_bool
5819     { \cs_set:Npn \arraystretch { 1 } }
5820     \dim_zero:N \extrarowheight
5821     #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).

```

5822     \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
5823     \bool_if:NTF \l_@@_NiceTabular_bool
5824     {
5825         \bool_lazy_all:nTF
5826         {
5827             { \int_compare:nNn { #2 } = 1 }
5828             { \dim_compare:n { \l_@@_col_width_dim >= \c_zero_dim } }
5829             { ! \l_@@_respect_arraystretch_bool }
5830         }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

5831     {
5832         \begin { minipage } [ \l_@@_vpos_of_block_tl ]
5833         { \l_@@_col_width_dim }
5834         \str_case:Vn \l_@@_hpos_block_str
5835         {
5836             c \centering
5837             r \raggedleft
5838             l \raggedright
5839         }
5840         #5
5841         \end { minipage }
5842     }
5843     {
5844         \use:x
5845         {
5846             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5847             { @ { } \l_@@_hpos_block_str @ { } }
5848         }
5849         #5
5850         \end { tabular }
5851     }
5852 }
5853 {
5854     \c_math_toggle_token
5855     \use:x
5856     {
5857         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5858         { @ { } \l_@@_hpos_block_str @ { } }
5859     }
5860     #5
5861     \end { array }
5862     \c_math_toggle_token
5863 }
5864 \group_end:
5865 }
5866 \bool_if:NT \g_@@_rotate_bool
5867 {
5868     \box_grotate:cn
5869     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5870     { 90 }
5871     \bool_gset_false:N \g_@@_rotate_bool
5872 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5873     \int_compare:nNnT { #2 } = 1
5874     {
5875         \dim_gset:Nn \g_@@_blocks_wd_dim

```

```

5876     {
5877         \dim_max:nn
5878         \g_@@_blocks_wd_dim
5879         {
5880             \box_wd:c
5881             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5882         }
5883     }
5884 }

```

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.

```

5885     \int_compare:nNnT { #1 } = 1
5886     {
5887         \dim_gset:Nn \g_@@_blocks_ht_dim
5888         {
5889             \dim_max:nn
5890             \g_@@_blocks_ht_dim
5891             {
5892                 \box_ht:c
5893                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5894             }
5895         }
5896         \dim_gset:Nn \g_@@_blocks_dp_dim
5897         {
5898             \dim_max:nn
5899             \g_@@_blocks_dp_dim
5900             {
5901                 \box_dp:c
5902                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5903             }
5904         }
5905     }
5906     \seq_gput_right:Nx \g_@@_blocks_seq
5907     {
5908         \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.

```

5909         { \exp_not:n { #3 } , \l_@@_hpos_block_str }
5910         {
5911             \box_use_drop:c
5912             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5913         }
5914     }
5915 }

```

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.

```

5916 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5917 {
5918     \seq_gput_right:Nx \g_@@_blocks_seq
5919     {
5920         \l_tmpa_tl
5921         { \exp_not:n { #3 } }
5922         \exp_not:n
5923         {
5924             {
5925                 \bool_if:NTF \l_@@_NiceTabular_bool
5926                 {

```

```

5927         \group_begin:
5928         \bool_if:NF \l_@@_respect_arraystretch_bool
5929         { \cs_set:Npn \arraystretch { 1 } }
5930         \dim_zero:N \extrarowheight
5931         #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).

```

5932         \bool_if:NT \g_@@_rotate_bool
5933         { \str_set:Nn \l_@@_hpos_block_str c }
5934         \use:x
5935         {
5936             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5937             { @ { } \l_@@_hpos_block_str @ { } }
5938         }
5939         #5
5940         \end { tabular }
5941         \group_end:
5942     }
5943 {
5944     \group_begin:
5945     \bool_if:NF \l_@@_respect_arraystretch_bool
5946     { \cs_set:Npn \arraystretch { 1 } }
5947     \dim_zero:N \extrarowheight
5948     #4
5949     \bool_if:NT \g_@@_rotate_bool
5950     { \str_set:Nn \l_@@_hpos_block_str c }
5951     \c_math_toggle_token
5952     \use:x
5953     {
5954         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5955         { @ { } \l_@@_hpos_block_str @ { } }
5956     }
5957     #5
5958     \end { array }
5959     \c_math_toggle_token
5960     \group_end:
5961 }
5962 }
5963 }
5964 }
5965 }

```

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).

```

5966 \keys_define:nn { NiceMatrix / Block / SecondPass }
5967 {
5968     tikz .code:n =
5969         \bool_if:NTF \c_@@_tikz_loaded_bool
5970         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
5971         { \@@_error:n { tikz-key-without~tikz } } ,
5972     tikz .value_required:n = true ,
5973     fill .tl_set:N = \l_@@_fill_tl ,
5974     fill .value_required:n = true ,
5975     draw .tl_set:N = \l_@@_draw_tl ,
5976     draw .default:n = default ,
5977     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5978     rounded-corners .default:n = 4 pt ,

```

```

5979   color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5980   color .value_required:n = true ,
5981   borders .clist_set:N = \l_@@_borders_clist ,
5982   borders .value_required:n = true ,
5983   hvlines .meta:n = { vlines , hlines } ,
5984   vlines .bool_set:N = \l_@@_vlines_block_bool ,
5985   vlines .default:n = true ,
5986   hlines .bool_set:N = \l_@@_hlines_block_bool ,
5987   hlines .default:n = true ,
5988   line-width .dim_set:N = \l_@@_line_width_dim ,
5989   line-width .value_required:n = true ,
5990   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5991   l .value_forbidden:n = true ,
5992   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5993   r .value_forbidden:n = true ,
5994   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5995   c .value_forbidden:n = true ,
5996   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
5997       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5998   L .value_forbidden:n = true ,
5999   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
6000       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6001   R .value_forbidden:n = true ,
6002   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
6003       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
6004   C .value_forbidden:n = true ,
6005   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
6006   t .value_forbidden:n = true ,
6007   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
6008   b .value_forbidden:n = true ,
6009   name .tl_set:N = \l_@@_block_name_str ,
6010   name .value_required:n = true ,
6011   name .initial:n = ,
6012   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6013   respect-arraystretch .default:n = true ,
6014   v-center .bool_set:N = \l_@@_v_center_bool ,
6015   v-center .default:n = true ,
6016   v-center .initial:n = false ,
6017   unknown .code:n = \@@_error:n { Unknown-key-for-Block }
6018 }

```

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.

```

6019 \cs_new_protected:Npn \@@_draw_blocks:
6020 {
6021   \cs_set_eq:NN \ialign \@@_old_ialign:
6022   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
6023 }
6024 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
6025 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

6026   \int_zero_new:N \l_@@_last_row_int
6027   \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.

```

6028 \int_compare:nNnTF { #3 } > { 99 }
6029 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
6030 { \int_set:Nn \l_@@_last_row_int { #3 } }
6031 \int_compare:nNnTF { #4 } > { 99 }
6032 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
6033 { \int_set:Nn \l_@@_last_col_int { #4 } }
6034 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6035 {
6036   \int_compare:nTF
6037   { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6038   {
6039     \msg_error:nnnn { nicematrix } { Block-too~large~2 } { #1 } { #2 }
6040     @@_msg_redirect_name:nn { Block-too~large~2 } { none }
6041     \group_begin:
6042     \globaldefs = 1
6043     @@_msg_redirect_name:nn { columns-not~used } { none }
6044     \group_end:
6045   }
6046   { \msg_error:nnnn { nicematrix } { Block-too~large~1 } { #1 } { #2 } }
6047 }
6048 {
6049   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6050   { \msg_error:nnnn { nicematrix } { Block-too~large~1 } { #1 } { #2 } }
6051   { @@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6052 }
6053 }
6054 \cs_new_protected:Npn @@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6055 {

```

The group is for the keys.

```

6056 \group_begin:
6057 \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }

```

We restrict the use of the key v-center to the case of a mono-row block.

```

6058 \bool_if:NT \l_@@_v_center_bool
6059 {
6060   \int_compare:nNnF { #1 } = { #3 }
6061   {
6062     @@_error:n { Wrong-use-of~v-center }
6063     \bool_set_false:N \l_@@_v_center_bool
6064   }
6065 }
6066 \bool_if:NT \l_@@_vlines_block_bool
6067 {
6068   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6069   {
6070     @@_vlines_block:nnn
6071     { \exp_not:n { #5 } }
6072     { #1 - #2 }
6073     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6074   }
6075 }
6076 \bool_if:NT \l_@@_hlines_block_bool
6077 {
6078   \tl_gput_right:Nx \g_nicematrix_code_after_tl
6079   {
6080     @@_hlines_block:nnn
6081     { \exp_not:n { #5 } }
6082     { #1 - #2 }
6083     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6084   }
6085 }
6086 \bool_if:nT

```

```

6087     { ! \l_@@_vlines_block_bool && ! \l_@@_hlines_block_bool }
6088     {

```

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).

```

6089     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6090     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6091   }
6092   \tl_if_empty:NF \l_@@_draw_tl
6093   {
6094     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6095     {
6096       \@@_stroke_block:nnn
6097       { \exp_not:n { #5 } }
6098       { #1 - #2 }
6099       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6100     }
6101     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6102     { { #1 } { #2 } { #3 } { #4 } }
6103   }
6104   \clist_if_empty:NF \l_@@_borders_clist
6105   {
6106     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6107     {
6108       \@@_stroke_borders_block:nnn
6109       { \exp_not:n { #5 } }
6110       { #1 - #2 }
6111       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6112     }
6113   }
6114   \tl_if_empty:NF \l_@@_fill_tl
6115   {

```

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`.

```

6116     \exp_last_unbraced:NV \@@_extract_brackets \l_@@_fill_tl \q_stop
6117     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6118     {
6119       \exp_not:N \roundedrectanglecolor
6120       [ \l_tmpa_tl ]
6121       { \exp_not:V \l_tmpb_tl }
6122       { #1 - #2 }
6123       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6124       { \dim_use:N \l_@@_rounded_corners_dim }
6125     }
6126   }
6127   \seq_if_empty:NF \l_@@_tikz_seq
6128   {
6129     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6130     {
6131       \@@_block_tikz:nnnnn
6132       { #1 }
6133       { #2 }
6134       { \int_use:N \l_@@_last_row_int }
6135       { \int_use:N \l_@@_last_col_int }
6136       { \seq_use:Nn \l_@@_tikz_seq { , } }
6137     }
6138   }
6139   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6140   {
6141     \tl_gput_right:Nx \g_@@_internal_code_after_tl

```

```

6142     {
6143         \@@_actually_diagbox:nnnnnn
6144         { #1 }
6145         { #2 }
6146         { \int_use:N \l_@@_last_row_int }
6147         { \int_use:N \l_@@_last_col_int }
6148         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6149     }
6150 }

6151 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6152 \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.

```

6153 \pgfpicture
6154 \pgfrememberpicturepositiononpagetrue
6155 \pgf@relevantforpicturesizefalse
6156 \@@_qpoint:n { row - #1 }
6157 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6158 \@@_qpoint:n { col - #2 }
6159 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6160 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
6161 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6162 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6163 \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.

```

6164 \@@_pgf_rect_node:nnnnn
6165 { \@@_env: - #1 - #2 - block }
6166 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6167 \str_if_empty:NF \l_@@_block_name_str
6168 {
6169     \pgfnodealias
6170     { \@@_env: - \l_@@_block_name_str }
6171     { \@@_env: - #1 - #2 - block }
6172     \str_if_empty:NF \l_@@_name_str
6173     {
6174         \pgfnodealias
6175         { \l_@@_name_str - \l_@@_block_name_str }
6176         { \@@_env: - #1 - #2 - block }
6177     }
6178 }

```


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.

```

6179     \bool_if:NF \l_@@_hpos_of_block_cap_bool
6180     {
6181         \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.

```

6182         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6183         {

```

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.

```

6184             \cs_if_exist:cT
6185             { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6186             {
6187                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6188                 {
6189                     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
6190                     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
6191                 }
6192             }
6193         }

```

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.

```

6194         \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
6195         {
6196             \@@_qpoint:n { col - #2 }
6197             \dim_set_eq:NN \l_tmpb_dim \pgf@x
6198         }
6199         \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
6200         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6201         {
6202             \cs_if_exist:cT
6203             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6204             {
6205                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6206                 {
6207                     \pgfpointanchor
6208                     { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6209                     { east }
6210                     \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
6211                 }
6212             }
6213         }
6214         \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
6215         {
6216             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
6217             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6218         }
6219         \@@_pgf_rect_node:nnnnn
6220         { \@@_env: - #1 - #2 - block - short }
6221         \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6222     }

```

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.

```

6223     \bool_if:NT \l_@@_medium_nodes_bool
6224     {
6225         \@@_pgf_rect_node:nnn

```

```

6226 { \@@_env: - #1 - #2 - block - medium }
6227 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
6228 {
6229   \pgfpointanchor
6230   { \@@_env:
6231     - \int_use:N \l_@@_last_row_int
6232     - \int_use:N \l_@@_last_col_int - medium
6233   }
6234   { south-east }
6235 }
6236 }

```

Now, we will put the label of the block beginning with the case of a \Block of one row.

```

6237 \bool_if:nTF
6238 { \int_compare_p:nNn { #1 } = { #3 } && ! \l_@@_v_center_bool }
6239 {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

6240 \int_compare:nNnTF { #1 } = 0
6241 { \l_@@_code_for_first_row_tl }
6242 {
6243   \int_compare:nNnT { #1 } = \l_@@_last_row_int
6244   \l_@@_code_for_last_row_tl
6245 }

```

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.

```

6246 \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in \pgf@x) the x -value of the center of the block.

```

6247 \pgfpointanchor
6248 {
6249   \@@_env: - #1 - #2 - block
6250   \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6251 }
6252 {
6253   \str_case:Vn \l_@@_hpos_block_str
6254   {
6255     c { center }
6256     l { west }
6257     r { east }
6258   }
6259 }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

6260 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
6261 \pgfset { inner~sep = \c_zero_dim }
6262 \pgfnode
6263 { rectangle }
6264 {
6265   \str_case:Vn \l_@@_hpos_block_str
6266   {
6267     c { base }
6268     l { base~west }
6269     r { base~east }
6270   }
6271 }
6272 { \box_use_drop:N \l_@@_cell_box } { } { }
6273 }

```

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).

```

6274 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

6275     \int_compare:nNnT { #2 } = 0
6276     { \str_set:Nn \l_@@_hpos_block_str r }
6277     \bool_if:nT \g_@@_last_col_found_bool
6278     {
6279         \int_compare:nNnT { #2 } = \g_@@_col_total_int
6280         { \str_set:Nn \l_@@_hpos_block_str l }
6281     }
6282     \pgftransformshift
6283     {
6284         \pgfpointanchor
6285         {
6286             \@@_env: - #1 - #2 - block
6287             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6288         }
6289         {
6290             \str_case:Vn \l_@@_hpos_block_str
6291             {
6292                 c { center }
6293                 l { west }
6294                 r { east }
6295             }
6296         }
6297     }
6298     \pgfset { inner-sep = \c_zero_dim }
6299     \pgfnode
6300     { rectangle }
6301     {
6302         \str_case:Vn \l_@@_hpos_block_str
6303         {
6304             c { center }
6305             l { west }
6306             r { east }
6307         }
6308     }
6309     { \box_use_drop:N \l_@@_cell_box } { } { }
6310 }
6311 \endpgfpicture
6312 \group_end:
6313 }

```

```

6314 \NewDocumentCommand \@@_extract_brackets { 0 { } }
6315 {
6316     \tl_set:Nn \l_tmpa_tl { #1 }
6317     \@@_store_in_tmpb_tl
6318 }
6319 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
6320 { \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).

```

6321 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
6322 {
6323     \group_begin:
6324     \tl_clear:N \l_@@_draw_tl
6325     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6326     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
6327     \pgfpicture
6328     \pgfrememberpicturepositiononpagetrue
6329     \pgf@relevantforpicturesizefalse
6330     \tl_if_empty:NF \l_@@_draw_tl

```

```
6331 {
```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```
6332 \str_if_eq:VnTF \l_@@_draw_tl { default }
6333 { \CT@arc@ }
6334 { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
6335 }
6336 \pgfsetcornersarced
6337 {
6338 \pgfpoint
6339 { \dim_use:N \l_@@_rounded_corners_dim }
6340 { \dim_use:N \l_@@_rounded_corners_dim }
6341 }
6342 \@@_cut_on_hyphen:w #2 \q_stop
6343 \bool_lazy_and:nnT
6344 { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
6345 { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
6346 {
6347 \@@_qpoint:n { row - \l_tmpa_tl }
6348 \dim_set:Nn \l_tmpb_dim { \pgf@y }
6349 \@@_qpoint:n { col - \l_tmpb_tl }
6350 \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
6351 \@@_cut_on_hyphen:w #3 \q_stop
6352 \int_compare:nNnT \l_tmpa_tl > \c@iRow
6353 { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
6354 \int_compare:nNnT \l_tmpb_tl > \c@jCol
6355 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
6356 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
6357 \dim_set:Nn \l_tmpa_dim { \pgf@y }
6358 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6359 \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
6360 \pgfpathrectanglecorners
6361 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6362 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6363 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```
6364 \pgfusepath { stroke }
6365 }
6366 \endpgfpicture
6367 \group_end:
6368 }
```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```
6369 \keys_define:nn { NiceMatrix / BlockStroke }
6370 {
6371 color .tl_set:N = \l_@@_draw_tl ,
6372 draw .tl_set:N = \l_@@_draw_tl ,
6373 draw .default:n = default ,
6374 line-width .dim_set:N = \l_@@_line_width_dim ,
6375 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6376 rounded-corners .default:n = 4 pt
6377 }
```

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).

```
6378 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
6379 {
6380 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6381 \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6382 \@@_cut_on_hyphen:w #2 \q_stop
6383 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
```

```

6384 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6385 \@@_cut_on_hyphen:w #3 \q_stop
6386 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6387 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6388 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
6389 {
6390   \use:x
6391   {
6392     \@@_vline:n
6393     {
6394       position = ##1 ,
6395       start = \l_@@_tmpc_tl ,
6396       end = \int_eval:n { \l_tmpa_tl - 1 }
6397     }
6398   }
6399 }
6400 }
6401 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
6402 {
6403   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6404   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6405   \@@_cut_on_hyphen:w #2 \q_stop
6406   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6407   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6408   \@@_cut_on_hyphen:w #3 \q_stop
6409   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6410   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6411   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
6412   {
6413     \use:x
6414     {
6415       \@@_hline:n
6416       {
6417         position = ##1 ,
6418         start = \l_@@_tmpd_tl ,
6419         end = \int_eval:n { \l_tmpb_tl - 1 }
6420       }
6421     }
6422   }
6423 }

```

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).

```

6424 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
6425 {
6426   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6427   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6428   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
6429   { \@@_error:n { borders~forbidden } }
6430   {
6431     \tl_clear_new:N \l_@@_borders_tikz_tl
6432     \keys_set:nV
6433     { NiceMatrix / OnlyForTikzInBorders }
6434     \l_@@_borders_clist
6435     \@@_cut_on_hyphen:w #2 \q_stop
6436     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6437     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6438     \@@_cut_on_hyphen:w #3 \q_stop
6439     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6440     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6441     \@@_stroke_borders_block_i:
6442   }

```

```

6443 }
6444 \hook_gput_code:nnn { begindocument } { . }
6445 {
6446   \cs_new_protected:Npx \@@_stroke_borders_block_i:
6447   {
6448     \c_@@_pgfortikzpicture_tl
6449     \@@_stroke_borders_block_ii:
6450     \c_@@_endpgfortikzpicture_tl
6451   }
6452 }
6453 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
6454 {
6455   \pgfrememberpicturepositiononpagetrue
6456   \pgf@relevantforpicturesizefalse
6457   \CT@arc@
6458   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
6459   \clist_if_in:NnT \l_@@_borders_clist { right }
6460   { \@@_stroke_vertical:n \l_tmpb_tl }
6461   \clist_if_in:NnT \l_@@_borders_clist { left }
6462   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
6463   \clist_if_in:NnT \l_@@_borders_clist { bottom }
6464   { \@@_stroke_horizontal:n \l_tmpa_tl }
6465   \clist_if_in:NnT \l_@@_borders_clist { top }
6466   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
6467 }
6468 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
6469 {
6470   tikz .code:n =
6471     \cs_if_exist:NTF \tikzpicture
6472     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
6473     { \@@_error:n { tikz-in-borders-without-tikz } } ,
6474   tikz .value_required:n = true ,
6475   top .code:n = ,
6476   bottom .code:n = ,
6477   left .code:n = ,
6478   right .code:n = ,
6479   unknown .code:n = \@@_error:n { bad-border }
6480 }

```

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).

```

6481 \cs_new_protected:Npn \@@_stroke_vertical:n #1
6482 {
6483   \@@_qpoint:n \l_@@_tmpc_tl
6484   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6485   \@@_qpoint:n \l_tmpa_tl
6486   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6487   \@@_qpoint:n { #1 }
6488   \tl_if_empty:NTF \l_@@_borders_tikz_tl
6489   {
6490     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
6491     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
6492     \pgfusepathqstroke
6493   }
6494   {
6495     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
6496     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
6497   }
6498 }

```

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).

```

6499 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
6500 {
6501   \@@_qpoint:n \l_@@_tmpd_tl
6502   \clist_if_in:NnTF \l_@@_borders_clist { left }
6503     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
6504     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
6505   \@@_qpoint:n \l_tmpb_tl
6506   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
6507   \@@_qpoint:n { #1 }
6508   \tl_if_empty:NTF \l_@@_borders_tikz_tl
6509     {
6510       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
6511       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6512       \pgfusepathqstroke
6513     }
6514     {
6515       \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
6516       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
6517     }
6518 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

6519 \keys_define:nn { NiceMatrix / BlockBorders }
6520 {
6521   borders .clist_set:N = \l_@@_borders_clist ,
6522   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6523   rounded-corners .default:n = 4 pt ,
6524   line-width .dim_set:N = \l_@@_line_width_dim ,
6525 }

```

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.

```

6526 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
6527 {
6528   \begin { tikzpicture }
6529   \clist_map_inline:nn { #5 }
6530     {
6531       \path [ ##1 ]
6532         ( #1 -| #2 )
6533         rectangle
6534         ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
6535     }
6536   \end { tikzpicture }
6537 }

```

How to draw the dotted lines transparently

```

6538 \cs_set_protected:Npn \@@_renew_matrix:
6539 {
6540   \RenewDocumentEnvironment { pmatrix } { } {
6541     { \pNiceMatrix }
6542     { \endpNiceMatrix }
6543   }
6544   \RenewDocumentEnvironment { vmatrix } { } {
6545     { \vNiceMatrix }
6546     { \endvNiceMatrix }
6547   }
6548   \RenewDocumentEnvironment { Vmatrix } { } {
6549     { \VNiceMatrix }
6550     { \endVNiceMatrix }
6551   }
6552   \RenewDocumentEnvironment { bmatrix } { } {
6553     { \bNiceMatrix }
6554     { \endbNiceMatrix }
6555   }
6556 }

```

```

6551     { \endbNiceMatrix }
6552 \RenewDocumentEnvironment { Bmatrix } { }
6553     { \BNiceMatrix }
6554     { \endBNiceMatrix }
6555 }

```

Automatic arrays

```

6556 \cs_new_protected:Npn \l_@@_set_size:n #1-#2 \q_stop
6557 {
6558     \int_set:Nn \l_@@_nb_rows_int { #1 }
6559     \int_set:Nn \l_@@_nb_cols_int { #2 }
6560 }

```

We will extract the potential keys l, r and c and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

6561 \keys_define:nn { NiceMatrix / Auto }
6562 {
6563     l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
6564     r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
6565     c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
6566 }
6567 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
6568 {
6569     \int_zero_new:N \l_@@_nb_rows_int
6570     \int_zero_new:N \l_@@_nb_cols_int
6571     \l_@@_set_size:n #4 \q_stop

```

The group is for the protection of `\l_@@_type_of_col_tl`.

```

6572 \group_begin:
6573 \tl_set:Nn \l_@@_type_of_col_tl c
6574 \keys_set_known:nn { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
6575 \use:x
6576 {
6577     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
6578     { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
6579     [ \exp_not:N \l_tmpa_tl ]
6580 }
6581 \int_compare:nNnT \l_@@_first_row_int = 0
6582 {
6583     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6584     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6585     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6586 }
6587 \prg_replicate:nn \l_@@_nb_rows_int
6588 {
6589     \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`).

```

6590     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
6591     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6592 }
6593 \int_compare:nNnT \l_@@_last_row_int > { -2 }
6594 {
6595     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6596     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6597     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6598 }
6599 \end { NiceArrayWithDelims }
6600 \group_end:
6601 }

```



```

6602 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6603 {
6604   \cs_set_protected:cpn { #1 AutoNiceMatrix }
6605   {
6606     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6607     \AutoNiceMatrixWithDelims { #2 } { #3 }
6608   }
6609 }
6610 \@@_define_com:nnn p ( )
6611 \@@_define_com:nnn b [ ]
6612 \@@_define_com:nnn v | |
6613 \@@_define_com:nnn V \| \|
6614 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

6615 \NewDocumentCommand \AutoNiceMatrix { 0 } { m 0 { } m ! 0 { } }
6616 {
6617   \group_begin:
6618   \bool_set_true:N \l_@@_NiceArray_bool
6619   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6620   \group_end:
6621 }

```

The redefinition of the command `\dotfill`

```

6622 \cs_set_eq:NN \@@_old_dotfill \dotfill
6623 \cs_new_protected:Npn \@@_dotfill:
6624 {

```

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}`).

```

6625   \@@_old_dotfill
6626   \bool_if:NT \l_@@_NiceTabular_bool
6627     { \group_insert_after:N \@@_dotfill_ii: }
6628     { \group_insert_after:N \@@_dotfill_i: }
6629   }
6630 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
6631 \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`.

```

6632 \cs_new_protected:Npn \@@_dotfill_iii:
6633 { \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.

```

6634 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6635 {
6636   \tl_gput_right:Nx \g_@@_internal_code_after_tl
6637   {
6638     \@@_actually_diagbox:nnnnnn
6639     { \int_use:N \c_iRow }
6640     { \int_use:N \c_jCol }
6641     { \int_use:N \c_iRow }
6642     { \int_use:N \c_jCol }
6643     { \exp_not:n { #1 } }
6644     { \exp_not:n { #2 } }
6645   }

```

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`.

```

6646   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6647   {
6648     { \int_use:N \c@iRow }
6649     { \int_use:N \c@jCol }
6650     { \int_use:N \c@iRow }
6651     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

6652     { }
6653   }
6654 }

```

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.

```

6655 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
6656 {
6657   \pgfpicture
6658   \pgf@relevantforpicturesizefalse
6659   \pgfrememberpicturepositiononpagetrue
6660   \@@_qpoint:n { row - #1 }
6661   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6662   \@@_qpoint:n { col - #2 }
6663   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6664   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6665   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
6666   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6667   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
6668   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6669   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6670   {

```

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.

```

6671   \CT@arc@
6672   \pgfsetroundcap
6673   \pgfusepathqstroke
6674 }
6675 \pgfset { inner~sep = 1 pt }
6676 \pgfscope
6677 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6678 \pgfnode { rectangle } { south-west }
6679 {
6680   \begin { minipage } { 20 cm }
6681   \@@_math_toggle_token: #5 \@@_math_toggle_token:
6682   \end { minipage }
6683 }
6684 { }
6685 { }
6686 \endpgfscope
6687 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6688 \pgfnode { rectangle } { north-east }
6689 {
6690   \begin { minipage } { 20 cm }
6691   \raggedleft
6692   \@@_math_toggle_token: #6 \@@_math_toggle_token:
6693   \end { minipage }
6694 }
6695 { }
6696 { }

```

```

6697 \endpgfpicture
6698 }

```

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.

```

6699 \keys_define:nn { NiceMatrix }
6700 {
6701   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6702   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6703 }
6704 \keys_define:nn { NiceMatrix / CodeAfter }
6705 {
6706   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6707   sub-matrix .value_required:n = true ,
6708   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6709   delimiters / color .value_required:n = true ,
6710   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6711   rules .value_required:n = true ,
6712   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
6713 }

```

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. 123.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

6714 \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 `\`.

```

6715 \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`.

```

6716 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
6717 {
6718   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6719   \@@_CodeAfter_iv:n
6720 }

```

We catch the argument of the command `\end` (in `#1`).

```

6721 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
6722 {

```

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.

```

6723   \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`.

```

6724   {
6725     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6726     \@@_CodeAfter_ii:n
6727   }
6728 }

```

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.

```
6729 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
6730 {
6731   \pgfpicture
6732   \pgfrememberpicturepositiononpagetrue
6733   \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.

```
6734   \@@_qpoint:n { row - 1 }
6735   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6736   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
6737   \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).

```
6738   \bool_if:nTF { #3 }
6739   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6740   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6741   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6742   {
6743     \cs_if_exist:cT
6744     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6745     {
6746       \pgfpointanchor
6747       { \@@_env: - ##1 - #2 }
6748       { \bool_if:nTF { #3 } { west } { east } }
6749       \dim_set:Nn \l_tmpa_dim
6750       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6751     }
6752   }
```

Now we can put the delimiter with a node of PGF.

```
6753   \pgfset { inner~sep = \c_zero_dim }
6754   \dim_zero:N \nulldelimiterspace
6755   \pgftransformshift
6756   {
6757     \pgfpoint
6758     { \l_tmpa_dim }
6759     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6760   }
6761   \pgfnode
6762   { rectangle }
6763   { \bool_if:nTF { #3 } { east } { west } }
6764   {
```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```
6765     \nullfont
6766     \c_math_toggle_token
6767     \tl_if_empty:NF \l_@@_delimiters_color_tl
6768     { \color { \l_@@_delimiters_color_tl } }
6769     \bool_if:nTF { #3 } { \left #1 } { \left . }
```

```

6770     \vcenter
6771     {
6772         \nullfont
6773         \hrule \@height
6774             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6775             \@depth \c_zero_dim
6776             \@width \c_zero_dim
6777     }
6778     \bool_if:nTF { #3 } { \right . } { \right #1 }
6779     \c_math_toggle_token
6780 }
6781 { }
6782 { }
6783 \endpgfpicture
6784 }

```

The command `\SubMatrix`

```

6785 \keys_define:nn { NiceMatrix / sub-matrix }
6786 {
6787     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6788     extra-height .value_required:n = true ,
6789     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6790     left-xshift .value_required:n = true ,
6791     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6792     right-xshift .value_required:n = true ,
6793     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6794     xshift .value_required:n = true ,
6795     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6796     delimiters / color .value_required:n = true ,
6797     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6798     slim .default:n = true ,
6799     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6800     hlines .default:n = all ,
6801     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6802     vlines .default:n = all ,
6803     hvlines .meta:n = { hlines, vlines } ,
6804     hvlines .value_forbidden:n = true ,
6805 }
6806 \keys_define:nn { NiceMatrix }
6807 {
6808     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6809     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6810     NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6811     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6812     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6813     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6814 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

6815 \keys_define:nn { NiceMatrix / SubMatrix }
6816 {
6817     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6818     delimiters / color .value_required:n = true ,
6819     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6820     hlines .default:n = all ,
6821     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6822     vlines .default:n = all ,
6823     hvlines .meta:n = { hlines, vlines } ,
6824     hvlines .value_forbidden:n = true ,
6825     name .code:n =
6826         \tl_if_empty:nTF { #1 }

```

```

6827 { \@@_error:n { Invalid-name-format } }
6828 {
6829   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6830   {
6831     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6832     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
6833     {
6834       \str_set:Nn \l_@@_submatrix_name_str { #1 }
6835       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6836     }
6837   }
6838   { \@@_error:n { Invalid-name-format } }
6839 } ,
6840 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6841 rules .value_required:n = true ,
6842 code .tl_set:N = \l_@@_code_tl ,
6843 code .value_required:n = true ,
6844 name .value_required:n = true ,
6845 unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
6846 }

6847 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
6848 {
6849   \peek_remove_spaces:n
6850   {
6851     \@@_cut_on_hyphen:w #3 \q_stop
6852     \tl_clear_new:N \l_@@_tmpc_tl
6853     \tl_clear_new:N \l_@@_tmpd_tl
6854     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6855     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6856     \@@_cut_on_hyphen:w #2 \q_stop
6857     \seq_gput_right:Nx \g_@@_submatrix_seq
6858     { { \l_tmpa_tl } { \l_tmpb_tl } { \l_@@_tmpc_tl } { \l_@@_tmpd_tl } }
6859     \tl_gput_right:Nn \g_@@_internal_code_after_tl
6860     { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6861   }
6862 }

```

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`.

```

6863 \hook_gput_code:nnn { begindocument } { . }
6864 {
6865   \tl_set:Nn \l_@@_argspec_tl { m m m m O { } E { _ ^ } { { } { } } }
6866   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
6867   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
6868   {
6869     \peek_remove_spaces:n
6870     {

```

```

6871         \@@_sub_matrix:nnnnnnn
6872         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
6873     }
6874 }
6875 }

```

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).

```

6876 \cs_new_protected:Npn \@@_compute_i_j:nn #1 #2
6877 {
6878     \tl_clear_new:N \l_@@_first_i_tl
6879     \tl_clear_new:N \l_@@_first_j_tl
6880     \tl_clear_new:N \l_@@_last_i_tl
6881     \tl_clear_new:N \l_@@_last_j_tl
6882     \@@_cut_on_hyphen:w #1 \q_stop
6883     \tl_if_eq:NnTF \l_tmpa_tl { last }
6884     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
6885     { \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl }
6886     \tl_if_eq:NnTF \l_tmpb_tl { last }
6887     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
6888     { \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl }
6889     \@@_cut_on_hyphen:w #2 \q_stop
6890     \tl_if_eq:NnTF \l_tmpa_tl { last }
6891     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
6892     { \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl }
6893     \tl_if_eq:NnTF \l_tmpb_tl { last }
6894     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
6895     { \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl }
6896 }
6897 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6898 {
6899     \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

6900     \@@_compute_i_j:nn { #2 } { #3 }
6901     \bool_lazy_or:nnTF
6902     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6903     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6904     { \@@_error:nn { Construct~too~large } { \SubMatrix } }
6905     {
6906         \str_clear_new:N \l_@@_submatrix_name_str
6907         \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6908         \pgfpicture
6909         \pgfrememberpicturepositiononpagetrue
6910         \pgf@relevantforpicturesizefalse
6911         \pgfset { inner~sep = \c_zero_dim }
6912         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6913         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

6914     \bool_if:NTF \l_@@_submatrix_slim_bool
6915     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6916     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6917     {
6918         \cs_if_exist:cT
6919         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6920         {
6921             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6922             \dim_set:Nn \l_@@_x_initial_dim
6923             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6924         }
6925         \cs_if_exist:cT

```

```

6926         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6927         {
6928             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6929             \dim_set:Nn \l_@@_x_final_dim
6930             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6931         }
6932     }
6933     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
6934     { \@@_error:nn { impossible~delimiter } { left } }
6935     {
6936         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
6937         { \@@_error:nn { impossible~delimiter } { right } }
6938         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
6939     }
6940     \endpgfpicture
6941 }
6942 \group_end:
6943 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

6944 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
6945 {
6946     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
6947     \dim_set:Nn \l_@@_y_initial_dim
6948     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
6949     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
6950     \dim_set:Nn \l_@@_y_final_dim
6951     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
6952     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
6953     {
6954         \cs_if_exist:cT
6955         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
6956         {
6957             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
6958             \dim_set:Nn \l_@@_y_initial_dim
6959             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6960         }
6961         \cs_if_exist:cT
6962         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6963         {
6964             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6965             \dim_set:Nn \l_@@_y_final_dim
6966             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6967         }
6968     }
6969     \dim_set:Nn \l_tmpa_dim
6970     {
6971         \l_@@_y_initial_dim - \l_@@_y_final_dim +
6972         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6973     }
6974     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

6975     \group_begin:
6976     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6977     \tl_if_empty:NF \l_@@_rules_color_tl
6978     { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6979     \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`.


```

6980 \seq_map_inline:Nn \g_@@_cols_vlism_seq
6981 {
6982   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6983   {
6984     \int_compare:nNnT
6985       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6986     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

6987       \@@_qpoint:n { col - ##1 }
6988       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6989       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6990       \pgfusepathqstroke
6991     }
6992   }
6993 }

```

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.

```

6994 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6995 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6996 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6997 {
6998   \bool_lazy_and:nnTF
6999     { \int_compare_p:nNn { ##1 } > 0 }
7000     {
7001       \int_compare_p:nNn
7002         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
7003     {
7004       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
7005       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7006       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
7007       \pgfusepathqstroke
7008     }
7009     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
7010 }

```

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.

```

7011 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
7012 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
7013 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
7014 {
7015   \bool_lazy_and:nnTF
7016     { \int_compare_p:nNn { ##1 } > 0 }
7017     {
7018       \int_compare_p:nNn
7019         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
7020     {
7021       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

7022 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

7023 \dim_set:Nn \l_tmpa_dim
7024 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7025 \str_case:nn { #1 }
7026 {
7027   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7028   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
7029   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
7030   }
7031   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

7032         \dim_set:Nn \l_tmpb_dim
7033         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7034         \str_case:nn { #2 }
7035         {
7036             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7037             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
7038             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
7039         }
7040         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7041         \pgfusepathqstroke
7042         \group_end:
7043     }
7044     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
7045 }

```

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).

```

7046     \str_if_empty:NF \l_@@_submatrix_name_str
7047     {
7048         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
7049         \l_@@_x_initial_dim \l_@@_y_initial_dim
7050         \l_@@_x_final_dim \l_@@_y_final_dim
7051     }
7052     \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`.

```

7053     \begin { pgfscope }
7054     \pgftransformshift
7055     {
7056         \pgfpoint
7057         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
7058         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7059     }
7060     \str_if_empty:NTF \l_@@_submatrix_name_str
7061     { \@@_node_left:nn #1 { } }
7062     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
7063     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

7064     \pgftransformshift
7065     {
7066         \pgfpoint
7067         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
7068         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
7069     }
7070     \str_if_empty:NTF \l_@@_submatrix_name_str
7071     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
7072     {
7073         \@@_node_right:nnnn #2
7074         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7075     }
7076     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7077     \flag_clear_new:n { nicematrix }
7078     \l_@@_code_tl
7079 }

```

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`.

```
7080 \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 `\@@_pgfpntanchor_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.

```
7081 \cs_new_protected:Npn \@@_pgfpntanchor:n #1
7082 {
7083   \use:e
7084   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpntanchor_i:nn #1 } }
7085 }
```

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 -.

```
7086 \cs_new:Npn \@@_pgfpntanchor_i:nn #1 #2
7087 { #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.

```
7088 \tl_const:Nn \c_@@_integers_alist_tl
7089 {
7090   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7091   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7092   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7093   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7094 }

7095 \cs_new:Npn \@@_pgfpntanchor_ii:w #1-#2\q_stop
7096 {
```

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`.

```
7097   \tl_if_empty:nTF { #2 }
7098   {
7099     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7100     {
7101       \flag_raise:n { nicematrix }
7102       \int_if_even:nTF { \flag_height:n { nicematrix } }
7103       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7104       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7105     }
7106     { #1 }
7107   }
```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```
7108   { \@@_pgfpntanchor_iii:w { #1 } #2 }
7109 }
```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpntanchor_i:nn`).

```
7110 \cs_new:Npn \@@_pgfpntanchor_iii:w #1 #2 -
```

```

7111 {
7112   \str_case:nnF { #1 }
7113   {
7114     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7115     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7116   }

```

Now the case of a node of the form $i-j$.

```

7117 {
7118   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7119   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7120 }
7121 }

```

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`).

```

7122 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7123 {
7124   \pgfnode
7125   { rectangle }
7126   { east }
7127   {
7128     \nullfont
7129     \c_math_toggle_token
7130     \tl_if_empty:NF \l_@@_delimiters_color_tl
7131     { \color { \l_@@_delimiters_color_tl } }
7132     \left #1
7133     \vcenter
7134     {
7135       \nullfont
7136       \hrule \@height \l_tmpa_dim
7137       \@depth \c_zero_dim
7138       \@width \c_zero_dim
7139     }
7140     \right .
7141     \c_math_toggle_token
7142   }
7143   { #2 }
7144   { }
7145 }

```

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.

```

7146 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
7147 {
7148   \pgfnode
7149   { rectangle }
7150   { west }
7151   {
7152     \nullfont
7153     \c_math_toggle_token
7154     \tl_if_empty:NF \l_@@_delimiters_color_tl
7155     { \color { \l_@@_delimiters_color_tl } }
7156     \left .
7157     \vcenter
7158     {
7159       \nullfont
7160       \hrule \@height \l_tmpa_dim
7161       \@depth \c_zero_dim
7162       \@width \c_zero_dim
7163     }

```

```

7164     \right #1
7165     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
7166     ^ { \smash { #4 } }
7167     \c_math_toggle_token
7168   }
7169   { #2 }
7170   { }
7171 }

```

Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

7172 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
7173 {
7174   \peek_remove_spaces:n
7175   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
7176 }
7177 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
7178 {
7179   \peek_remove_spaces:n
7180   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
7181 }
7182 \keys_define:nn { NiceMatrix / Brace }
7183 {
7184   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
7185   left-shorten .default:n = true ,
7186   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
7187   shorten .meta:n = { left-shorten , right-shorten } ,
7188   right-shorten .default:n = true ,
7189   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
7190   yshift .value_required:n = true ,
7191   yshift .initial:n = \c_zero_dim ,
7192   color .tl_set:N = \l_tmpa_tl ,
7193   color .value_required:n = true ,
7194   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
7195 }

```

#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*.

```

7196 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
7197 {
7198   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

7199   \@@_compute_i_j:nn { #1 } { #2 }
7200   \bool_lazy_or:nnTF
7201   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7202   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7203   {
7204     \str_if_eq:nnTF { #5 } { under }
7205     { \@@_error:nn { Construct~too~large } { \UnderBrace } }
7206     { \@@_error:nn { Construct~too~large } { \OverBrace } }
7207   }
7208   {
7209     \tl_clear:N \l_tmpa_tl % added the 2022-02-25
7210     \keys_set:nn { NiceMatrix / Brace } { #4 }
7211     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } } % added the 2022-02-25
7212     \pgfpicture

```

```

7213 \pgfrememberpicturepositiononpagetrue
7214 \pgf@relevantforpicturesizefalse
7215 \bool_if:NT \l_@@_brace_left_shorten_bool
7216 {
7217     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7218     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7219     {
7220         \cs_if_exist:cT
7221         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7222         {
7223             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7224             \dim_set:Nn \l_@@_x_initial_dim
7225             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7226         }
7227     }
7228 }
7229 \bool_lazy_or:nnT
7230 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
7231 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
7232 {
7233     \@@_qpoint:n { col - \l_@@_first_j_tl }
7234     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7235 }
7236 \bool_if:NT \l_@@_brace_right_shorten_bool
7237 {
7238     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
7239     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7240     {
7241         \cs_if_exist:cT
7242         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7243         {
7244             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7245             \dim_set:Nn \l_@@_x_final_dim
7246             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7247         }
7248     }
7249 }
7250 \bool_lazy_or:nnT
7251 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
7252 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
7253 {
7254     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
7255     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7256 }
7257 \pgfset { inner-sep = \c_zero_dim }
7258 \str_if_eq:nnTF { #5 } { under }
7259 { \@@_underbrace_i:n { #3 } }
7260 { \@@_overbrace_i:n { #3 } }
7261 \endpgfpicture
7262 }
7263 \group_end:
7264 }

```

The argument is the text to put above the brace.

```

7265 \cs_new_protected:Npn \@@_overbrace_i:n #1
7266 {
7267     \@@_qpoint:n { row - \l_@@_first_i_tl }
7268     \pgftransformshift
7269     {
7270         \pgfpoint
7271         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7272         { \pgf@y + \l_@@_brace_yshift_dim }
7273     }
7274     \pgfnode

```

```

7275 { rectangle }
7276 { south }
7277 {
7278   \vbox_top:n
7279   {
7280     \group_begin:
7281     \everycr { }
7282     \halign
7283     {
7284       \hfil ## \hfil \crrc
7285       \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7286       \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7287       \hbox_to_wd:nn
7288       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7289       { \downbracefill } \cr
7290     }
7291     \group_end:
7292   }
7293 }
7294 { }
7295 { }
7296 }

```

The argument is the text to put under the brace.

```

7297 \cs_new_protected:Npn \@@_underbrace_i:n #1
7298 {
7299   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
7300   \pgftransformshift
7301   {
7302     \pgfpoint
7303     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7304     { \pgf@y - \l_@@_brace_yshift_dim }
7305   }
7306   \pgfnode
7307   { rectangle }
7308   { north }
7309   {
7310     \group_begin:
7311     \everycr { }
7312     \vbox:n
7313     {
7314       \halign
7315       {
7316         \hfil ## \hfil \crrc
7317         \hbox_to_wd:nn
7318         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7319         { \upbracefill } \cr
7320         \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7321         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7322       }
7323     }
7324     \group_end:
7325   }
7326   { }
7327   { }
7328 }

```

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.

```
7329 \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.

```
7330 \bool_new:N \c_@@_footnote_bool
```

```
7331 \@@_msg_new:nnn { Unknown-key-for-package }
```

```
7332 {
7333   The~key~'\l_keys_key_str'~is-unknown. \\
7334   If~you-go-on,~it~will~be~ignored. \\
7335   For~a~list~of~the~available~keys,~type~H<return>.
7336 }
```

```
7337 {
7338   The~available~keys~are~(in~alphabetic~order):~
7339   allow-letter-for-dotted-lines,~
7340   footnote,~
7341   footnotehyper,~
7342   renew-dots,~and
7343   renew-matrix.
7344 }
```

```
7345 \keys_define:nn { NiceMatrix / Package }
```

```
7346 {
7347   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
7348   renew-dots .value_forbidden:n = true ,
7349   renew-matrix .code:n = \@@_renew_matrix: ,
7350   renew-matrix .value_forbidden:n = true ,
7351   transparent .code:n = \@@_fatal:n { Key-transparent } ,
7352   transparent .value_forbidden:n = true,
7353   footnote .bool_set:N = \c_@@_footnote_bool ,
7354   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
7355   allow-letter-for-dotted-lines .code:n =
7356   {
7357     \group_begin:
7358     \globaldefs = 1
7359     \@@_msg_redirect_name:nn { letter-for-dotted-lines } { none }
7360     \group_end:
7361   } ,
7362   allow-letter-for-dotted-lines .value_forbidden:n = true ,
7363   unknown .code:n = \@@_error:n { Unknown-key-for-package }
7364 }
```

```
7365 \ProcessKeysOptions { NiceMatrix / Package }
```

```
7366 \@@_msg_new:nn { footnote-with-footnotehyper-package }
```

```
7367 {
7368   You~can't~use~the~option~'footnote'~because~the~package~
7369   footnotehyper~has~already~been~loaded.~
7370   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
7371   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7372   of~the~package~footnotehyper.\\
7373   If~you~go~on,~the~package~footnote~won't~be~loaded.
7374 }
```

```
7375 \@@_msg_new:nn { footnotehyper-with-footnote-package }
```

```
7376 {
7377   You~can't~use~the~option~'footnotehyper'~because~the~package~
7378   footnote~has~already~been~loaded.~
7379   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
7380   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
```



```

7381 of~the~package~footnote.\\
7382 If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
7383 }

```

```

7384 \bool_if:NT \c_@@_footnote_bool
7385 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

7386 \@ifclassloaded { beamer }
7387 { \bool_set_false:N \c_@@_footnote_bool }
7388 {
7389 \@ifpackageloaded { footnotehyper }
7390 { \@@_error:n { footnote~with~footnotehyper~package } }
7391 { \usepackage { footnote } }
7392 }
7393 }
7394 \bool_if:NT \c_@@_footnotehyper_bool
7395 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

7396 \@ifclassloaded { beamer }
7397 { \bool_set_false:N \c_@@_footnote_bool }
7398 {
7399 \@ifpackageloaded { footnote }
7400 { \@@_error:n { footnotehyper~with~footnote~package } }
7401 { \usepackage { footnotehyper } }
7402 }
7403 \bool_set_true:N \c_@@_footnote_bool
7404 }

```

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

```

7405 \seq_new:N \g_@@_types_of_matrix_seq
7406 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
7407 {
7408 NiceMatrix ,
7409 pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
7410 }
7411 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
7412 { \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`.

```

7413 \cs_new_protected:Npn \@@_error_too_much_cols:
7414 {
7415 \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
7416 {
7417 \int_compare:nNnTF \l_@@_last_col_int = { -2 }
7418 { \@@_fatal:n { too~much~cols~for~matrix } }
7419 {
7420 \bool_if:NF \l_@@_last_col_without_value_bool
7421 { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
7422 }
7423 }

```

```

7424     { \@@_fatal:n { too-much-cols-for-array } }
7425 }

```

The following command must *not* be protected since it's used in an error message.

```

7426 \cs_new:Npn \@@_message_hdotsfor:
7427 {
7428   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
7429   { ~Maybe-you-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
7430 }
7431 \@@_msg_new:nn { negative-weight }
7432 {
7433   The-weight-of-the-'X'~columns-must-be-positive-and-you-have-used-
7434   the-value-'#1'.~If-you-go-on,~the-absolute-value-will-be-used.
7435 }
7436 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
7437 {
7438   You-try-to-use-more-columns-than-allowed-by-your~
7439   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-
7440   columns-is-\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
7441   exterior~columns).~This-error-is-fatal.
7442 }
7443 \@@_msg_new:nn { too-much-cols-for-matrix }
7444 {
7445   You-try-to-use-more-columns-than-allowed-by-your~
7446   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that~the-maximal~
7447   number-of-columns-for-a-matrix-is-fixed-by-the-LaTeX-counter~
7448   'MaxMatrixCols'.~Its-actual-value-is-\int_use:N \c@MaxMatrixCols.~
7449   This-error-is-fatal.
7450 }

```

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`.

```

7451 \@@_msg_new:nn { too-much-cols-for-array }
7452 {
7453   You-try-to-use-more-columns-than-allowed-by-your~
7454   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
7455   \int_use:N \g_@@_static_num_of_col_int\
7456   ~(plus~the~potential~exterior~ones).~
7457   This-error-is-fatal.
7458 }
7459 \@@_msg_new:nn { hvlines-except-corners }
7460 {
7461   The-key~'hvlines-except-corners'~is-now-obsolete.~You-should~instead-use-the~
7462   keys~'hvlines'~and~'corners'.\\
7463   However,~you-can-go-on~for~this~time.~This-message~won't-be-shown~anymore~
7464   in~this~document.
7465 }
7466 \@@_msg_new:nn { last-col-not-used }
7467 {
7468   The-key~'last-col'~is~in~force~but~you-have-not-used~that~last~column~
7469   in~your~\@@_full_name_env:~.~However,~you-can-go-on.
7470 }
7471 \@@_msg_new:nn { columns-not-used }
7472 {
7473   The-preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7474   \g_@@_static_num_of_col_int\ columns~but~you-use-only~\int_use:N \c@jCol.\\
7475   You-can-go-on~but~the~columns~you~did~not~used~won't-be-created.
7476 }
7477 \@@_msg_new:nn { in-first-col }
7478 {
7479   You-can't-use-the-command~#1 in~the~first~column~(number~0)~of~the~array.\\
7480   If-you-go-on,~this~command~will~be~ignored.

```

```

7481 }
7482 \@@_msg_new:nn { in~last~col }
7483 {
7484     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
7485     If~you~go~on,~this~command~will~be~ignored.
7486 }
7487 \@@_msg_new:nn { in~first~row }
7488 {
7489     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
7490     If~you~go~on,~this~command~will~be~ignored.
7491 }
7492 \@@_msg_new:nn { in~last~row }
7493 {
7494     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
7495     If~you~go~on,~this~command~will~be~ignored.
7496 }
7497 \@@_msg_new:nn { double~closing~delimiter }
7498 {
7499     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
7500     delimiter.~This~delimiter~will~be~ignored.
7501 }
7502 \@@_msg_new:nn { delimiter~after~opening }
7503 {
7504     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
7505     delimiter.~This~delimiter~will~be~ignored.
7506 }
7507 \@@_msg_new:nn { bad~option~for~line~style }
7508 {
7509     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
7510     is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
7511 }
7512 \@@_msg_new:nnn { Unknown~key~for~custom~line }
7513 {
7514     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
7515     If~you~go~on,~it~will~be~ignored. \\
7516     For~a~list~of~the~available~keys,~type~H<return>.
7517 }
7518 {
7519     The~available~keys~are~(in~alphabetic~order):~
7520     color,~
7521     command,~
7522     dotted,~
7523     letter,~
7524     multiplicity,~
7525     sep~color,~
7526     tikz,~and~width.
7527 }
7528 \@@_msg_new:nn { Unknown~key~for~xdots }
7529 {
7530     As~for~now,~there~is~only~three~keys~available~here:~'color',~'line~style'~
7531     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
7532     this~key~will~be~ignored.
7533 }
7534 \@@_msg_new:nn { Unknown~key~for~rowcolors }
7535 {
7536     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
7537     (and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
7538     this~key~will~be~ignored.
7539 }

```

```

7540 \@@_msg_new:nn { ampersand~in~light-syntax }
7541 {
7542     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
7543     ~you~have~used~the~key~'light-syntax'.~This~error~is~fatal.
7544 }
7545 \@@_msg_new:nn { Construct~too~large }
7546 {
7547     Your~command~\token_to_str:N #1
7548     can't~be~drawn~because~your~matrix~is~too~small.\\
7549     If~you~go~on,~this~command~will~be~ignored.
7550 }
7551 \@@_msg_new:nn { double-backslash~in~light-syntax }
7552 {
7553     You~can't~use~\token_to_str:N \\~to~separate~rows~because~you~have~used~
7554     the~key~'light-syntax'.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
7555     (set~by~the~key~'end-of-row').~This~error~is~fatal.
7556 }
7557 \@@_msg_new:nn { standard~cline~in~document }
7558 {
7559     The~key~'standard~cline'~is~available~only~in~the~preamble.\\
7560     If~you~go~on~this~command~will~be~ignored.
7561 }
7562 \@@_msg_new:nn { bad~value~for~baseline }
7563 {
7564     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
7565     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
7566     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
7567     If~you~go~on,~a~value~of~1~will~be~used.
7568 }
7569 \@@_msg_new:nn { Invalid~name~format }
7570 {
7571     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
7572     \SubMatrix.\\
7573     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
7574     If~you~go~on,~this~key~will~be~ignored.
7575 }
7576 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
7577 {
7578     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
7579     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
7580     number~is~not~valid.~If~you~go~on,~it~will~be~ignored.
7581 }
7582 \@@_msg_new:nn { impossible~delimiter }
7583 {
7584     It's~impossible~to~draw~the~#1~delimiter~of~your~
7585     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
7586     in~that~column.
7587     \bool_if:NT \l_@@_submatrix_slim_bool
7588     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
7589     If~you~go~on,~this~\token_to_str:N \SubMatrix\ will~be~ignored.
7590 }
7591 \@@_msg_new:nn { width~without~X~columns }
7592 {
7593     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column. \\
7594     If~you~go~on,~that~key~will~be~ignored.
7595 }
7596 \@@_msg_new:nn { empty~environment }
7597 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }
7598 \@@_msg_new:nn { Wrong~use~of~v~center }

```

```

7599 {
7600     You~should~not~use~the~key~'v-center'~here~because~your~block~is~not~
7601     mono~row.~However,~you~can~go~on.
7602 }

7603 \@@_msg_new:nn { No~letter~and~no~command }
7604 {
7605     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
7606     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~key~'command'~
7607     (to~draw~horizontal~rules).\\
7608     However,~you~can~go~on.
7609 }

7610 \@@_msg_new:nn { letter~for~dotted~lines }
7611 {
7612     The~key~'letter~for~dotted~lines'~is~now~obsolete~(you~should~
7613     use~'custom~line'~instead).~However,~you~can~go~on~for~this~time.~
7614     If~you~don't~want~to~see~that~message~again,~you~should~
7615     load~'nicematrix'~with~the~key~'allow~letter~for~dotted~lines'.~
7616     However,~'letter~for~dotted~lines'~will~be~deleted~in~a~future~
7617     version~of~'nicematrix'.
7618 }

7619 \@@_msg_new:nn { Forbidden~letter }
7620 {
7621     You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line\\.\\
7622     If~you~go~on,~it~will~be~ignored.
7623 }

7624 \@@_msg_new:nn { key~width~without~key~tikz }
7625 {
7626     In~'custom~line',~you~have~used~'width'~without~'tikz'.~That's~not~correct.~
7627     If~you~go~on,~that~key~'width'~will~be~discarded.
7628 }

7629 \@@_msg_new:nn { Several~letters }
7630 {
7631     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~
7632     have~used~'\l_@@_letter_str').\\
7633     If~you~go~on,~it~will~be~ignored.
7634 }

7635 \@@_msg_new:nn { Delimiter~with~small }
7636 {
7637     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
7638     because~the~key~'small'~is~in~force\\.\\
7639     This~error~is~fatal.
7640 }

7641 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
7642 {
7643     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code~after'~
7644     can't~be~executed~because~a~cell~doesn't~exist\\.\\
7645     If~you~go~on~this~command~will~be~ignored.
7646 }

7647 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
7648 {
7649     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\\
7650     in~this~\@@_full_name_env:\\
7651     If~you~go~on,~this~key~will~be~ignored\\.\\
7652     For~a~list~of~the~names~already~used,~type~H~<return>.
7653 }
7654 {
7655     The~names~already~defined~in~this~\@@_full_name_env:\\ are:~
7656     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
7657 }

7658 \@@_msg_new:nn { r~or~l~with~preamble }

```

```

7659 {
7660   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
7661   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
7662   your~\@@_full_name_env:~\
7663   If~you~go~on,~this~key~will~be~ignored.
7664 }

7665 \@@_msg_new:nn { Hdotsfor~in~col-0 }
7666 {
7667   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
7668   the~array.~This~error~is~fatal.
7669 }

7670 \@@_msg_new:nn { bad~corner }
7671 {
7672   #1~is~an~incorrect~specification~for~a~corner~(in~the~keys~
7673   'corners'~and~'except-corners').~The~available~
7674   values~are:~NW,~SW,~NE~and~SE.\
7675   If~you~go~on,~this~specification~of~corner~will~be~ignored.
7676 }

7677 \@@_msg_new:nn { bad~border }
7678 {
7679   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
7680   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
7681   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
7682   also~use~the~key~'tikz'
7683   \bool_if:nF \c_@@_tikz_loaded_bool
7684   {~if~you~load~the~LaTeX~package~'tikz'}).\
7685   If~you~go~on,~this~specification~of~border~will~be~ignored.
7686 }

7687 \@@_msg_new:nn { tikz~key~without~tikz }
7688 {
7689   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
7690   \Block'~because~you~have~not~loaded~Tikz.~
7691   If~you~go~on,~this~key~will~be~ignored.
7692 }

7693 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
7694 {
7695   In~the~\@@_full_name_env:,~you~must~use~the~key~
7696   'last~col'~without~value.\
7697   However,~you~can~go~on~for~this~time~
7698   (the~value~'\l_keys_value_tl'~will~be~ignored).
7699 }

7700 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
7701 {
7702   In~\NiceMatrixoptions,~you~must~use~the~key~
7703   'last~col'~without~value.\
7704   However,~you~can~go~on~for~this~time~
7705   (the~value~'\l_keys_value_tl'~will~be~ignored).
7706 }

7707 \@@_msg_new:nn { Block~too~large~1 }
7708 {
7709   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
7710   too~small~for~that~block. \
7711 }

7712 \@@_msg_new:nn { Block~too~large~2 }
7713 {
7714   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7715   \g_@@_static_num_of_col_int\
7716   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
7717   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
7718   (&)~at~the~end~of~the~first~row~of~your~

```

```

7719 \@@_full_name_env:.\
7720 If~you~go~on,this~block~and~maybe~others~will~be~ignored.
7721 }
7722 \@@_msg_new:nn { unknown~column~type }
7723 {
7724   The~column~type~'#1'~in~your~\@@_full_name_env:\
7725   is~unknown. \
7726   This~error~is~fatal.
7727 }
7728 \@@_msg_new:nn { colon~without~arydshln }
7729 {
7730   The~column~type~': '~in~your~\@@_full_name_env:\
7731   is~unknown.~If~you~want~to~use~': '~of~arydshln',~you~should~
7732   load~that~package.~If~you~want~a~dotted~line~of~'nicematrix',~you~
7733   should~use~'\l_@@_letter_for_dotted_lines_str'.\
7734   This~error~is~fatal.
7735 }
7736 \@@_msg_new:nn { tabularnote~forbidden }
7737 {
7738   You~can't~use~the~command~\token_to_str:N\tabularnote\
7739   ~in~a~\@@_full_name_env:..~This~command~is~available~only~in~
7740   \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \
7741   If~you~go~on,~this~command~will~be~ignored.
7742 }
7743 \@@_msg_new:nn { borders~forbidden }
7744 {
7745   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
7746   because~the~option~'rounded~corners'~
7747   is~in~force~with~a~non~zero~value.\
7748   If~you~go~on,~this~key~will~be~ignored.
7749 }
7750 \@@_msg_new:nn { bottomrule~without~booktabs }
7751 {
7752   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
7753   loaded~'booktabs'.\
7754   If~you~go~on,~this~key~will~be~ignored.
7755 }
7756 \@@_msg_new:nn { enumitem~not~loaded }
7757 {
7758   You~can't~use~the~command~\token_to_str:N\tabularnote\
7759   ~because~you~haven't~loaded~'enumitem'.\
7760   If~you~go~on,~this~command~will~be~ignored.
7761 }
7762 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
7763 {
7764   You~have~used~the~key~'tikz'~in~the~definition~of~a~
7765   customized~line~(with~'custom~line')~but~Tikz~is~not~loaded.~
7766   You~can~go~on~but~you~will~have~another~error~if~you~actually~
7767   use~that~custom~line.
7768 }
7769 \@@_msg_new:nn { tikz~in~borders~without~tikz }
7770 {
7771   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
7772   command~\token_to_str:N\Block')~but~Tikz~is~not~loaded.~
7773   If~you~go~on,~that~key~will~be~ignored.
7774 }
7775 \@@_msg_new:nn { color~in~custom~line~with~tikz }
7776 {
7777   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
7778   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~

```

```

7779     If~you~go~on,~the~key~'color'~will~be~discarded.
7780 }

7781 \@@_msg_new:nn { Wrong~last~row }
7782 {
7783     You~have~used~'last~row'~\int_use:N \l_@@_last_row_int'~but~your~
7784     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
7785     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
7786     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
7787     without~value~(more~compilations~might~be~necessary).
7788 }

7789 \@@_msg_new:nn { Yet~in~env }
7790 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }

7791 \@@_msg_new:nn { Outside~math~mode }
7792 {
7793     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
7794     (and~not~in~\token_to_str:N \vcenter).\\
7795     This~error~is~fatal.
7796 }

7797 \@@_msg_new:nn { One~letter~allowed }
7798 {
7799     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
7800     If~you~go~on,~it~will~be~ignored.
7801 }

7802 \@@_msg_new:nn { varwidth~not~loaded }
7803 {
7804     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
7805     loaded.\\
7806     If~you~go~on,~your~column~will~behave~like~'p'.
7807 }

7808 \@@_msg_new:nnn { Unknown~key~for~Block }
7809 {
7810     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
7811     \Block.\\ If~you~go~on,~it~will~be~ignored. \\
7812     For~a~list~of~the~available~keys,~type~H<return>.
7813 }
7814 {
7815     The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
7816     hlines,~hvlines,~l,~line~width,~name,~rounded~corners,~r,~respect~arraystretch,
7817     t,~tikz~and~vlines.
7818 }

7819 \@@_msg_new:nn { Version~of~siunitx~too~old }
7820 {
7821     You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
7822     is~too~old.~You~need~at~least~v3.0.\\
7823     This~error~is~fatal.
7824 }

7825 \@@_msg_new:nnn { Unknown~key~for~Brace }
7826 {
7827     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
7828     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
7829     If~you~go~on,~it~will~be~ignored. \\
7830     For~a~list~of~the~available~keys,~type~H<return>.
7831 }
7832 {
7833     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
7834     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
7835     right~shorten)~and~yshift.
7836 }

7837 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
7838 {

```



```

7839 The~key~'\l_keys_key_str'~is~unknown.\\
7840 If~you~go~on,~it~will~be~ignored. \\
7841 For~a~list~of~the~available~keys~in~\token_to_str:N
7842 \CodeAfter,~type~H~<return>.
7843 }
7844 {
7845 The~available~keys~are~(in~alphabetic~order):~
7846 delimiters/color,~
7847 rules~(with~the~subkeys~'color'~and~'width'),~
7848 sub-matrix~(several~subkeys)~
7849 and~xdots~(several~subkeys).~
7850 The~latter~is~for~the~command~\token_to_str:N \line.
7851 }
7852 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
7853 {
7854 The~key~'\l_keys_key_str'~is~unknown.\\
7855 If~you~go~on,~this~key~will~be~ignored. \\
7856 For~a~list~of~the~available~keys~in~\token_to_str:N
7857 \SubMatrix,~type~H~<return>.
7858 }
7859 {
7860 The~available~keys~are~(in~alphabetic~order):~
7861 'delimiters/color',~
7862 'extra-height',~
7863 'hlines',~
7864 'hvlines',~
7865 'left-xshift',~
7866 'name',~
7867 'right-xshift',~
7868 'rules'~(with~the~subkeys~'color'~and~'width'),~
7869 'slim',~
7870 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
7871 and~'right-xshift').\\
7872 }
7873 \@@_msg_new:nnn { Unknown~key~for~notes }
7874 {
7875 The~key~'\l_keys_key_str'~is~unknown.\\
7876 If~you~go~on,~it~will~be~ignored. \\
7877 For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
7878 }
7879 {
7880 The~available~keys~are~(in~alphabetic~order):~
7881 bottomrule,~
7882 code-after,~
7883 code-before,~
7884 detect-duplicates,~
7885 enumitem-keys,~
7886 enumitem-keys-para,~
7887 para,~
7888 label-in-list,~
7889 label-in-tabular~and~
7890 style.
7891 }
7892 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
7893 {
7894 The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7895 \token_to_str:N \RowStyle. \\
7896 If~you~go~on,~it~will~be~ignored. \\
7897 For~a~list~of~the~available~keys,~type~H~<return>.
7898 }
7899 {
7900 The~available~keys~are~(in~alphabetic~order):~
7901 'bold',~

```

```

7902     'cell-space-top-limit',~
7903     'cell-space-bottom-limit',~
7904     'cell-space-limits',~
7905     'color',~
7906     'nb-rows'~and~
7907     'rowcolor'.
7908 }
7909 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
7910 {
7911     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7912     \token_to_str:N \NiceMatrixOptions. \\
7913     If~you~go~on,~it~will~be~ignored. \\
7914     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7915 }
7916 {
7917     The~available~keys~are~(in~alphabetic~order):~
7918     allow~duplicate~names,~
7919     cell-space-bottom-limit,~
7920     cell-space-limits,~
7921     cell-space-top-limit,~
7922     code-for-first-col,~
7923     code-for-first-row,~
7924     code-for-last-col,~
7925     code-for-last-row,~
7926     corners,~
7927     custom-key,~
7928     create-extra-nodes,~
7929     create-medium-nodes,~
7930     create-large-nodes,~
7931     delimiters~(several~subkeys),~
7932     end-of-row,~
7933     first-col,~
7934     first-row,~
7935     hlines,~
7936     hvlines,~
7937     last-col,~
7938     last-row,~
7939     left-margin,~
7940     light-syntax,~
7941     notes~(several~subkeys),~
7942     nullify-dots,~
7943     renew-dots,~
7944     renew-matrix,~
7945     respect-arraystretch,~
7946     right-margin,~
7947     rules~(with~the~subkeys~'color'~and~'width'),~
7948     small,~
7949     sub-matrix~(several~subkeys),~
7950     vlimes,~
7951     xdots~(several~subkeys).
7952 }
7953 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
7954 {
7955     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7956     \{NiceArray\}. \\
7957     If~you~go~on,~it~will~be~ignored. \\
7958     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7959 }
7960 {
7961     The~available~keys~are~(in~alphabetic~order):~
7962     b,~
7963     baseline,~
7964     c,~

```

```

7965 cell-space-bottom-limit,~
7966 cell-space-limits,~
7967 cell-space-top-limit,~
7968 code-after,~
7969 code-for-first-col,~
7970 code-for-first-row,~
7971 code-for-last-col,~
7972 code-for-last-row,~
7973 colortbl-like,~
7974 columns-width,~
7975 corners,~
7976 create-extra-nodes,~
7977 create-medium-nodes,~
7978 create-large-nodes,~
7979 delimiters/color,~
7980 extra-left-margin,~
7981 extra-right-margin,~
7982 first-col,~
7983 first-row,~
7984 hlines,~
7985 hvlines,~
7986 last-col,~
7987 last-row,~
7988 left-margin,~
7989 light-syntax,~
7990 name,~
7991 notes/bottomrule,~
7992 notes/para,~
7993 nullify-dots,~
7994 renew-dots,~
7995 respect-arraystretch,~
7996 right-margin,~
7997 rules~(with~the~subkeys~'color'~and~'width'),~
7998 small,~
7999 t,~
8000 tabularnote,~
8001 vlines,~
8002 xdots/color,~
8003 xdots/shorten~and~
8004 xdots/line-style.
8005 }

```

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).

```

8006 \@@_msg_new:nnn { Unknown-key-for-NiceMatrix }
8007 {
8008   The-key~'\l_keys_key_str'~is-unknown~for~the~
8009   \@@_full_name_env:. \\\
8010   If-you-go-on,~it~will~be~ignored. \\\
8011   For-a-list-of-the~*principal*~available-keys,~type-H~<return>.
8012 }
8013 {
8014   The~available~keys~are~(in~alphabetic~order):~
8015   b,~
8016   baseline,~
8017   c,~
8018   cell-space-bottom-limit,~
8019   cell-space-limits,~
8020   cell-space-top-limit,~
8021   code-after,~
8022   code-for-first-col,~
8023   code-for-first-row,~
8024   code-for-last-col,~
8025   code-for-last-row,~

```

```

8026 colortbl-like,~
8027 columns-width,~
8028 corners,~
8029 create-extra-nodes,~
8030 create-medium-nodes,~
8031 create-large-nodes,~
8032 delimiters~(several~subkeys),~
8033 extra-left-margin,~
8034 extra-right-margin,~
8035 first-col,~
8036 first-row,~
8037 hlines,~
8038 hvlines,~
8039 l,~
8040 last-col,~
8041 last-row,~
8042 left-margin,~
8043 light-syntax,~
8044 name,~
8045 nullify-dots,~
8046 r,~
8047 renew-dots,~
8048 respect-arraystretch,~
8049 right-margin,~
8050 rules~(with~the~subkeys~'color'~and~'width'),~
8051 small,~
8052 t,~
8053 vlimes,~
8054 xdots/color,~
8055 xdots/shorten~and~
8056 xdots/line-style.
8057 }
8058 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
8059 {
8060   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
8061   \{NiceTabular\}. \\
8062   If~you~go~on,~it~will~be~ignored. \\
8063   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
8064 }
8065 {
8066   The~available~keys~are~(in~alphabetic~order):~
8067   b,~
8068   baseline,~
8069   c,~
8070   cell-space-bottom-limit,~
8071   cell-space-limits,~
8072   cell-space-top-limit,~
8073   code-after,~
8074   code-for-first-col,~
8075   code-for-first-row,~
8076   code-for-last-col,~
8077   code-for-last-row,~
8078   colortbl-like,~
8079   columns-width,~
8080   corners,~
8081   custom-line,~
8082   create-extra-nodes,~
8083   create-medium-nodes,~
8084   create-large-nodes,~
8085   extra-left-margin,~
8086   extra-right-margin,~
8087   first-col,~
8088   first-row,~

```

```

8089 hlines,~
8090 hvlines,~
8091 last-col,~
8092 last-row,~
8093 left-margin,~
8094 light-syntax,~
8095 name,~
8096 notes/bottomrule,~
8097 notes/para,~
8098 nullify-dots,~
8099 renew-dots,~
8100 respect-arraystretch,~
8101 right-margin,~
8102 rules~(with~the~subkeys~'color'~and~'width'),~
8103 t,~
8104 tabularnote,~
8105 vlines,~
8106 xdots/color,~
8107 xdots/shorten~and~
8108 xdots/line-style.
8109 }

8110 \@@_msg_new:nnn { Duplicate~name }
8111 {
8112   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
8113   the~same~environment~name~twice.~You~can~go~on,~but,~
8114   maybe,~you~will~have~incorrect~results~especially~
8115   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
8116   message~again,~use~the~key~'allow-duplicate-names'~in~
8117   '\token_to_str:N \NiceMatrixOptions'.\\
8118   For~a~list~of~the~names~already~used,~type~H~<return>. \\
8119 }
8120 {
8121   The~names~already~defined~in~this~document~are:~
8122   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
8123 }

8124 \@@_msg_new:nn { Option~auto~for~columns~width }
8125 {
8126   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
8127   If~you~go~on,~the~key~will~be~ignored.
8128 }

```

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`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

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

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 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.
Option `allow-duplicate-names`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`).
Possibility to draw vertical dotted lines to separate columns with the specifier “:” in the preamble (similar to the classical specifier “|” and the specifier “:” of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “:” in the preamble.
Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.
Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of “|”) as `\hdotsfor` does.
Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.
Error message when the user gives an incorrect value for `last-row`.
A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol “:” (in the preamble of the array) and `\line` in `code-after`).
The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.
The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by `colortbl`.
Correction of a bug: it was not possible to use the colon “:” in the preamble of an array when `pdflatex` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.
The option `columns-width=auto` doesn’t need any more a second compilation.
The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).
The previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX 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`

Changes between version 6.7 and 6.8

In the notes of a tabular (with the command `\tabularnote`), the duplicates are now detected: when several commands `\tabularnote` are used with the same argument, only one note is created at the end of the tabular (but all the labels are present, of course).

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>	1247, 5741
<code>\@@_Block_i</code>	5746, 5747, 5751
<code>\@@_Block_ii:nnnnn</code>	5751, 5752
<code>\@@_Block_iv:nnnnn</code>	5789, 5793
<code>\@@_Block_iv:nnnnnn</code>	6022, 6024
<code>\@@_Block_v:nnnnn</code>	5790, 5916
<code>\@@_Block_v:nnnnnn</code>	6051, 6054
<code>\@@_Cdots</code>	1168, 1239, 4013
<code>\g_@@_Cdots_lines_tl</code>	1270, 3204
<code>\@@_CodeAfter:</code>	1251, 6714
<code>\@@_CodeAfter_i:</code>	901, 2851, 2896, 6715
<code>\@@_CodeAfter_ii:n</code> ..	6714, 6715, 6716, 6726
<code>\@@_CodeAfter_iv:n</code>	6719, 6721
<code>\@@_CodeAfter_keys:</code>	3145, 3164
<code>\@@_CodeBefore:w</code>	1412, 1414
<code>\@@_CodeBefore_keys:</code>	1392, 1409
<code>\@@_Ddots</code>	1170, 1241, 4045
<code>\g_@@_Ddots_lines_tl</code>	1273, 3202
<code>\g_@@_Hvdotsfor_lines_tl</code>	1275, 3200, 4129, 4206, 7428
<code>\@@_Hdotsfor:</code>	1173, 1245, 4105
<code>\@@_Hdotsfor:nnnn</code>	4131, 4144
<code>\@@_Hdotsfor_i</code>	4114, 4120, 4127
<code>\@@_Hline:</code>	1243, 5181
<code>\@@_Hline_i:n</code>	5181, 5182, 5188
<code>\@@_Hline_ii:nn</code>	5185, 5188
<code>\@@_Hline_iii:n</code>	5186, 5189
<code>\@@_Hspace:</code>	1244, 4099
<code>\@@_Iddots</code>	1171, 1242, 4069
<code>\g_@@_Iddots_lines_tl</code>	1274, 3203
<code>\@@_Ldots</code>	1167, 1172, 1238, 3997
<code>\g_@@_Ldots_lines_tl</code>	1271, 3205
<code>\l_@@_Matrix_bool</code>	244, 1687, 1711, 2447, 2789, 2795, 2803, 2964
<code>\l_@@_NiceArray_bool</code>	241, 409, 1316, 1555, 1627, 1750, 1757, 1769, 2447, 2777, 2789, 2795, 2803, 2940, 4972, 4976, 5127, 5137, 5167, 5171, 6618
<code>\g_@@_NiceMatrixBlock_int</code>	224, 5499, 5504, 5507, 5518
<code>\l_@@_NiceTabular_bool</code> ...	181, 242, 906, 1094, 1391, 1394, 1522, 1660, 1664, 1758, 1770, 2778, 2996, 3016, 3025, 5823, 5925, 6626
<code>\@@_NotEmpty:</code>	1253, 2988
<code>\@@_OnlyMainNiceMatrix:n</code>	1249, 4759
<code>\@@_OnlyMainNiceMatrix_i:n</code>	4762, 4769, 4772

<code>\@@_OverBrace</code>	3139, 7177	<code>\l_@@_block_name_str</code>	6009, 6090, 6167, 6170, 6175
<code>\@@_RowStyle:n</code>	1254, 4357	<code>\@@_block_tikz:nnnnn</code>	6131, 6526
<code>\@@_S:</code>	212, 1800	<code>\g_@@_blocks_dp_dim</code>	235, 987, 990, 991, 5896, 5899
<code>\@@_SubMatrix</code>	3137, 6867	<code>\g_@@_blocks_ht_dim</code>	234, 993, 996, 997, 5887, 5890
<code>\@@_SubMatrix_in_code_before</code>	1384, 6847	<code>\g_@@_blocks_seq</code>	292, 1537, 2457, 5906, 5918, 6022
<code>\@@_UnderBrace</code>	3138, 7172	<code>\g_@@_blocks_wd_dim</code>	233, 981, 984, 985, 5875, 5878
<code>\@@_V:</code>	1715, 1796	<code>\c_@@_booktabs_loaded_bool</code>	34, 35, 1187, 2490
<code>\@@_Vdots</code>	1169, 1240, 4029	<code>\l_@@_borders_clist</code>	310, 5981, 6104, 6434, 6459, 6461, 6463, 6465, 6502, 6521
<code>\g_@@_Vdots_lines_tl</code>	1272, 3201	<code>\l_@@_borders_tikz_tl</code>	6431, 6472, 6488, 6495, 6508, 6515
<code>\@@_Vdotsfor:</code>	1246, 4204	<code>\@@_brace:nnnnn</code>	7175, 7180, 7196
<code>\@@_Vdotsfor:nnnn</code>	4208, 4219	<code>\l_@@_brace_left_shorten_bool</code>	7184, 7215, 7230
<code>\@@_W:</code>	1714, 1799, 2304	<code>\l_@@_brace_right_shorten_bool</code>	7186, 7236, 7251
<code>\@@_X</code>	1808, 3011	<code>\l_@@_brace_yshift_dim</code>	7189, 7272, 7304
<code>\l_@@_X_column_bool</code>	246, 2196, 5787	<code>\@@_cartesian_color:nn</code>	4457, 4469, 4478, 4600
<code>\l_@@_X_columns_aux_bool</code>	275, 1580, 2185	<code>\@@_cartesian_path:</code>	4461, 4699
<code>\l_@@_X_columns_dim</code>	276, 1581, 1590, 1596, 2188	<code>\@@_cartesian_path:n</code>	4509, 4641, 4699
<code>\@@_actually_color:</code>	1393, 4444	<code>\@@_cell_begin:w</code>	898, 1837, 1967, 2038, 2069, 2195, 2313, 2334, 2355, 2973
<code>\@@_actually_diagbox:nnnnnn</code>	5800, 6143, 6638, 6655	<code>\l_@@_cell_box</code>	905, 953, 955, 961, 967, 970, 974, 983, 984, 989, 990, 995, 996, 1007, 1008, 1009, 1010, 1012, 1015, 1019, 1021, 1040, 1055, 1057, 1064, 1065, 1078, 1189, 1300, 1302, 1995, 1999, 2003, 2006, 2037, 2048, 2198, 2354, 2365, 2854, 2878, 2881, 2883, 2900, 2923, 2927, 6151, 6272, 6309, 6633
<code>\@@_actually_draw_Cdots:</code>	3570, 3574	<code>\@@_cell_end:</code>	1000, 1839, 1986, 2043, 2074, 2204, 2315, 2344, 2360, 2973
<code>\@@_actually_draw_Ddots:</code>	3720, 3724	<code>\l_@@_cell_space_bottom_limit_dim</code>	504, 584, 1010, 4403
<code>\@@_actually_draw_Iddots:</code>	3768, 3772	<code>\l_@@_cell_space_top_limit_dim</code>	503, 582, 1008, 4392
<code>\@@_actually_draw_Ldots:</code>	3531, 3535, 4195	<code>\@@_cellcolor</code>	1375, 4511, 4523, 4524, 4730
<code>\@@_actually_draw_Vdots:</code>	3652, 3656, 4270	<code>\@@_cellcolor_tabular</code>	1177, 4724
<code>\@@_add_to_colors_seq:nn</code>	4430, 4442, 4443, 4467, 4476, 4485, 4494, 4598	<code>\g_@@_cells_seq</code>	2634, 2635, 2636, 2638
<code>\@@_adjust_pos_of_blocks_seq:</code>	3121, 3166	<code>\@@_center_cell_box:</code>	1945, 1990
<code>\@@_adjust_pos_of_blocks_seq_i:nnnnn</code>	3169, 3171	<code>\@@_chessboardcolors</code>	1383, 4516
<code>\@@_adjust_size_box:</code>	979, 1006, 2047, 2364, 2875, 2920	<code>\@@_cline</code>	154, 1237
<code>\@@_adjust_to_submatrix:nn</code>	3415, 3518, 3557, 3638, 3713, 3761	<code>\@@_cline_i:nn</code>	155, 156, 176, 179
<code>\@@_adjust_to_submatrix:nnnnnn</code>	3422, 3424	<code>\@@_cline_i:w</code>	156, 157
<code>\@@_after_array:</code>	1696, 3029	<code>\@@_cline_ii:w</code>	161, 163
<code>\g_@@_after_col_zero_bool</code>	277, 1135, 2852, 4111	<code>\@@_cline_iii:w</code>	160, 164, 165
<code>\@@_analyze_end:Nn</code>	2595, 2641	<code>\l_@@_code_before_bool</code>	281, 674, 701, 1108, 1307, 1338, 1552, 2655, 2672, 2690, 2721, 2747, 2784, 2823, 3158
<code>\l_@@_argspec_tl</code>	22, 3995, 3996, 3997, 4013, 4029, 4045, 4069, 4125, 4126, 4127, 4202, 4203, 4204, 4280, 4281, 4282, 6865, 6866, 6867	<code>\g_@@_code_before_tl</code>	1541, 1550, 1553, 3155
<code>\@@_array:</code>	1092, 2596, 2623	<code>\l_@@_code_before_tl</code>	280, 673, 1337, 1392, 1553
<code>\@@_arraycolor</code>	1381, 4529	<code>\l_@@_code_for_first_col_tl</code>	601, 2865
<code>\c_@@_arydshln_loaded_bool</code>	31, 32	<code>\l_@@_code_for_first_row_tl</code>	605, 917, 6241
<code>\l_@@_auto_columns_width_bool</code>	529, 684, 2708, 2712, 5494	<code>\l_@@_code_for_last_col_tl</code>	603, 2909
<code>\g_@@_aux_found_bool</code>	1287, 1292, 1416, 1543, 1546	<code>\l_@@_code_for_last_row_tl</code>	607, 924, 6244
<code>\g_@@_aux_tl</code>	1549, 1578, 1703, 3038, 3052, 3060, 3153, 5381	<code>\l_@@_code_tl</code>	268, 6842, 7078
<code>\l_@@_bar_at_end_of_pream_bool</code>	1751, 1886, 2781	<code>\l_@@_col_max_int</code>	304, 3281, 3292, 3360, 3420, 3437
<code>\l_@@_baseline_tl</code>	518, 519, 677, 678, 679, 680, 1101, 1629, 2400, 2412, 2417, 2419, 2424, 2429, 2519, 2520, 2524, 2529, 2531, 2536		
<code>\@@_begin_of_NiceMatrix:nn</code>	2962, 2976, 2984		
<code>\@@_begin_of_row:</code>	904, 929, 1198, 2853		
<code>\l_@@_block_auto_columns_width_bool</code>	1535, 2713, 5487, 5492, 5502, 5512		
<code>\g_@@_block_box_int</code>	322, 1516, 5795, 5809, 5811, 5869, 5881, 5893, 5902, 5912		

\l_@@_col_min_int	4678, 4708, 4709, 6342, 6351, 6382, 6385,
..... 303, 3286, 3349, 3354, 3418, 3435	6405, 6408, 6435, 6438, 6851, 6856, 6882, 6889
\g_@@_col_total_int	\g_@@_ddots_int 3100, 3740, 3741
..... 229, 1023, 1266, 1346, 1357,	\@@_def_env:nnn
1429, 1613, 2285, 2286, 2740, 2741, 2771, 2946, 2957, 2958, 2959, 2960, 2961
2826, 2831, 2838, 2840, 2899, 3033, 3035,	\@@_define_com:nnn
3047, 3605, 3623, 3683, 4265, 4749, 5545, 6602, 6610, 6611, 6612, 6613, 6614
5555, 5589, 5676, 6034, 6279, 6903, 6952, 7202	\@@_define_h_custom_line:nn 5260, 5284, 5297
\l_@@_col_width_dim	\@@_delimiter:nnn
..... 226, 227, 1966, 2036, 5828, 5833	... 2102, 2123, 2131, 2144, 2150, 2156, 6729
\@@_color_index:n 4588, 4609, 4612	\l_@@_delimiters_color_tl 542,
\l_@@_color_int	748, 1405, 1652, 1653, 1670, 1671, 6708,
..... 4552, 4553, 4570, 4571, 4591, 4602	6767, 6768, 6795, 6817, 7130, 7131, 7154, 7155
\l_@@_color_tl	\l_@@_delimiters_max_width_bool
312, 4585, 4586, 4596, 4599, 5736, 5813, 5815 543, 746, 1675
\g_@@_colors_seq 1388, 4433, 4437, 4438, 4448	\g_@@_delta_x_one_dim 3102, 3743, 3753
\l_@@_colors_seq 4547, 4548, 4592, 4611, 4613	\g_@@_delta_x_two_dim 3104, 3791, 3801
\@@_colortbl_like: 1175, 1257	\g_@@_delta_y_one_dim 3103, 3745, 3753
\l_@@_colortbl_like_bool 501, 700, 1257, 1739	\g_@@_delta_y_two_dim 3105, 3793, 3801
\l_@@_colortbl_loaded_bool . 108, 112, 1206	\@@_diagbox:nn 1252, 6634
\l_@@_cols_tl	\@@_dotfill: 6623
4460, 4508, 4539, 4549, 4550, 4600, 4647, 4650	\@@_dotfill_i: 6628, 6630
\g_@@_cols_vlism_seq	\@@_dotfill_ii: 6627, 6630, 6631
..... 255, 1305, 1734, 1818, 6980	\@@_dotfill_iii: 6631, 6632
\@@_columncolor 1382, 4472	\l_@@_dotted_boa 4877
\@@_columncolor_preamble 1179, 4747	\l_@@_dotted_bool
\c_@@_columncolor_regex 67, 1742 330, 3813, 4791, 4879, 5058, 5060
\l_@@_columns_width_dim	\@@_double_int_eval:n 4276, 4290, 4291
..... 225, 685, 810, 2709, 2715, 5500, 5506	\g_@@_dp_ante_last_row_dim 932, 1222
\g_@@_com_or_env_str 259, 260, 263	\g_@@_dp_last_row_dim
\l_@@_command_str 932, 933, 1225, 1226, 1301, 1302, 1646
..... 5209, 5230, 5250, 5271, 5286, 5295	\g_@@_dp_row_zero_dim
\@@_computations_for_large_nodes: 952, 953, 1216, 1217, 1639, 2513, 2552
..... 5616, 5629, 5634	\@@_draw_Cdots:nnn 3555
\@@_computations_for_medium_nodes: ..	\@@_draw_Ddots:nnn 3711
..... 5536, 5605, 5615, 5626	\@@_draw_Iddots:nnn 3759
\@@_compute_a_corner:nnnnnn 5369, 5371, 5373, 5375, 5388	\@@_draw_Ldots:nnn 3516
\@@_compute_corners: 3120, 5361	\@@_draw_Vdots:nnn 3636
\@@_compute_i_j:nn 6876, 6900, 7199	\@@_draw_blocks: 2457, 6019
\@@_construct_preamble: 1313, 1708	\@@_draw_dotted_lines: 3119, 3189
\l_@@_corners_cells_seq	\@@_draw_dotted_lines_i: 3192, 3196
..... 296, 4644, 4684, 4855, 4861, 4868,	\l_@@_draw_first_bool . 319, 4060, 4084, 4095
5036, 5042, 5049, 5363, 5379, 5383, 5384, 5444	\@@_draw_hlines: 3122, 5163
\l_@@_corners_clist	\@@_draw_line: 3553,
..... 523, 662, 667, 4829, 5010, 5364	3598, 3709, 3757, 3805, 3807, 4329, 4948, 5143
\@@_create_blocks_nodes: 1366, 1461	\@@_draw_line_ii:nn 4309, 4313
\@@_create_col_nodes: 2599, 2627, 2647	\@@_draw_line_iii:nn 4316, 4320
\@@_create_diag_nodes: ... 1363, 3068, 3222	\@@_draw_standard_dotted_line: . 3814, 3846
\@@_create_extra_nodes: .. 1459, 2456, 5526	\@@_draw_standard_dotted_line_i: 3909, 3913
\@@_create_large_nodes: 5534, 5610	\l_@@_draw_tl 308, 5975,
\@@_create_medium_and_large_nodes: ..	5979, 6092, 6324, 6330, 6332, 6334, 6371, 6372
..... 5531, 5621	\@@_draw_unstandard_dotted_line: 3815, 3817
\@@_create_medium_nodes: 5532, 5600	\@@_draw_unstandard_dotted_line:n ...
\@@_create_nodes: 5607, 5618, 5628, 5631, 5672 3820, 3823, 3830
\@@_create_one_block_node:nnnnn 1467, 1470	\@@_draw_unstandard_dotted_line:nnn ..
\@@_create_row_node: 1104, 1138, 1188 3825, 3831, 3845
\@@_custom_line:n 575, 5207, 5316	\@@_draw_vlines: 3123, 4968
\l_@@_custom_line_commands_seq	\g_@@_empty_cell_bool .. 289, 1014, 1024,
..... 272, 1255, 5295	2888, 2935, 4011, 4027, 4043, 4067, 4090, 4101
\@@_custom_line_i:n 5244, 5298	\l_@@_end_int 4783,
\@@_cut_on_hyphen:w 342,	4807, 4808, 4819, 4846, 4989, 4990, 5000, 5027
355, 4501, 4506, 4566, 4654, 4655, 4677,	\l_@@_end_of_row_tl
 539, 540, 595, 2619, 2620, 7554

```

\c_@@_endpgfortikzpicture_tl .....
..... 47, 52, 3193, 4317, 6450
\c_@@_enumitem_loaded_bool .....
..... 37, 38, 382, 726, 735
\@@_env: ..... 219, 223, 938, 944,
1041, 1047, 1061, 1069, 1114, 1120, 1126,
1201, 1347, 1348, 1353, 1354, 1359, 1360,
1372, 1426, 1427, 1432, 1435, 1438, 1441,
1450, 1451, 1456, 1457, 1483, 2656, 2659,
2661, 2677, 2683, 2686, 2695, 2701, 2704,
2726, 2732, 2735, 2753, 2759, 2765, 2792,
2801, 2815, 2826, 2831, 2840, 3073, 3074,
3079, 3080, 3088, 3094, 3132, 3239, 3241,
3248, 3250, 3251, 3256, 3259, 3321, 3389,
3454, 3465, 3478, 3481, 3500, 3503, 3608,
3611, 3626, 3629, 4158, 4176, 4233, 4251,
4302, 4304, 4323, 4326, 5399, 5418, 5436,
5558, 5560, 5568, 5679, 5688, 5706, 6165,
6170, 6171, 6176, 6185, 6189, 6203, 6208,
6220, 6226, 6227, 6230, 6249, 6286, 6744,
6747, 6919, 6921, 6926, 6928, 6955, 6957,
6962, 6964, 7062, 7074, 7221, 7223, 7242, 7244
\g_@@_env_int .....
. 218, 219, 221, 1534, 1544, 1547, 1702, 5715
\@@_error:n ..... 12, 385, 410,
552, 564, 617, 657, 742, 800, 809, 816, 823,
831, 848, 855, 863, 864, 865, 871, 876, 877,
878, 892, 894, 895, 896, 1407, 1574, 1608,
1618, 1690, 2027, 2434, 2495, 2541, 4355,
4542, 5219, 5221, 5226, 5231, 5239, 5248,
5282, 5971, 6017, 6062, 6429, 6473, 6479,
6712, 6827, 6838, 6845, 7194, 7363, 7390, 7400
\@@_error:nn 13, 14, 692, 2105, 2151, 2157,
2181, 4000, 4003, 4016, 4019, 4032, 4035,
4049, 4050, 4055, 4056, 4073, 4074, 4079,
4080, 5377, 6832, 6904, 6934, 6937, 7205, 7206
\@@_error:nnn ..... 15, 4307, 7009, 7044
\@@_error_too_much_cols: ..... 1779, 7413
\@@_everycr: ..... 1131, 1211, 1214
\@@_everycr_i: ..... 1131, 1132
\l_@@_except_borders_bool .....
..... 534, 629, 4972, 4976, 5167, 5171
\@@_exec_code_before: ..... 1307, 1386
\@@_expand_clist:N ..... 347, 1185, 1186
\@@_expand_clist:NN ..... 4647, 4648, 4700
\l_@@_exterior_arraycolsep_bool .....
..... 520, 806, 1760, 1772, 2780
\l_@@_extra_left_margin_dim .....
..... 537, 645, 1329, 2886
\l_@@_extra_right_margin_dim .....
..... 538, 646, 1569, 2931, 3686
\@@_extract_brackets ..... 6116, 6314
\@@_extract_coords_values: .... 5697, 5704
\@@_fatal:n .....
.. 16, 251, 1525, 2079, 2083, 2112, 2604,
2608, 2610, 2644, 4116, 7351, 7418, 7421, 7424
\@@_fatal:nn ..... 17, 1829, 2307
\l_@@_fill_tl ..... 307, 5973, 6114, 6116
\l_@@_final_i_int .....
..... 3109, 3268, 3273, 3276, 3301,
3309, 3313, 3322, 3330, 3410, 3466, 3547,
3620, 3626, 3629, 4149, 4177, 4245, 4255, 4257
\l_@@_final_j_int .....
3110, 3269, 3274, 3281, 3286, 3292, 3302,
3310, 3314, 3323, 3331, 3409, 3411, 3467,
3500, 3503, 3511, 4170, 4180, 4182, 4224, 4253
\l_@@_final_open_bool .....
3112, 3275, 3279, 3282, 3289, 3295, 3299,
3315, 3544, 3579, 3584, 3595, 3659, 3669,
3674, 3695, 3732, 3780, 3917, 3932, 3963,
3964, 4147, 4171, 4183, 4222, 4246, 4258, 4299
\@@_find_extremities_of_line:nnnn ...
..... 3263, 3521, 3560, 3641, 3716, 3764
\l_@@_first_col_int .....
..... 142, 155, 333, 334, 597, 869,
904, 1357, 1429, 1622, 1752, 2650, 2670,
3045, 3605, 3623, 4109, 4668, 4761, 5545,
5555, 5589, 5637, 5676, 6583, 6589, 6595, 6952
\l_@@_first_i_tl ..... 6878,
6884, 6885, 6915, 6946, 6955, 6957, 7012,
7019, 7021, 7103, 7114, 7118, 7218, 7239, 7267
\l_@@_first_j_tl .....
6879, 6887, 6888, 6919, 6921, 6982, 6995,
7002, 7004, 7104, 7115, 7119, 7221, 7223, 7233
\l_@@_first_row_int .. 331, 332, 598, 873,
1264, 1351, 1424, 1637, 2431, 2510, 2538,
2549, 3042, 3475, 3497, 5538, 5552, 5579,
5636, 5674, 6182, 6200, 6581, 6741, 6916, 7565
\c_@@_footnote_bool .....
1511, 1706, 7330, 7353, 7384, 7387, 7397, 7403
\c_@@_footnotehyper_bool . 7329, 7354, 7394
\c_@@_forbidden_letters_str .... 5238, 5264
\@@_full_name_env: .....
261, 7439, 7446, 7454, 7469, 7473, 7579,
7597, 7637, 7650, 7655, 7660, 7662, 7695,
7714, 7719, 7724, 7730, 7739, 7784, 7793, 8009
\@@_hdottedline: ..... 5473
\@@_hdottedline:n ..... 5481, 5483
\@@_hdottedline_i: ..... 5476, 5478
\@@_hline:n 4986, 5178, 5198, 5292, 5484, 6415
\@@_hline_i: ..... 4992, 4995
\@@_hline_ii: ..... 5020, 5028, 5056
\@@_hline_iii: ..... 5064, 5068
\@@_hline_iv: ..... 5061, 5114
\@@_hline_v: ..... 5065, 5146
\@@_hlines_block:nnn ..... 6080, 6401
\l_@@_hlines_block_bool 321, 5986, 6076, 6087
\l_@@_hlines_clist .. 326, 609, 623, 628,
664, 1139, 1141, 1145, 1185, 3122, 5176, 5177
\l_@@_hpos_block_str .... 314, 315, 5720,
5722, 5724, 5726, 5728, 5730, 5767, 5768,
5770, 5822, 5834, 5847, 5858, 5909, 5933,
5937, 5950, 5955, 5990, 5992, 5994, 5996,
5999, 6002, 6253, 6265, 6276, 6280, 6290, 6302
\l_@@_hpos_cell_str ..... 231, 232,
1837, 1933, 1935, 2039, 2313, 2356, 5766, 5768
\l_@@_hpos_col_str .....
.... 1889, 1891, 1893, 1895, 1920, 1932,
1936, 1938, 1946, 1947, 1950, 1955, 2022, 2173
\l_@@_hpos_of_block_cap_bool .....
.... 316, 5997, 6000, 6003, 6179, 6250, 6287
\g_@@_ht_last_row_dim .....
..... 934, 1223, 1224, 1299, 1300, 1645
\g_@@_ht_row_one_dim .. 960, 961, 1220, 1221

```



```

\g_@@_ht_row_zero_dim .....
.... 954, 955, 1218, 1219, 1640, 2512, 2551
\@@_i: ..... 5538, 5540,
5541, 5542, 5543, 5552, 5558, 5560, 5561,
5562, 5563, 5568, 5569, 5570, 5571, 5579,
5582, 5584, 5585, 5586, 5638, 5640, 5643,
5644, 5648, 5649, 5674, 5679, 5681, 5683,
5687, 5688, 5699, 5706, 5708, 5710, 5714, 5715
\g_@@_iddots_int ..... 3101, 3788, 3789
\l_@@_in_env_bool .... 238, 409, 1525, 1526
\l_@@_initial_i_int ..... 3107,
3266, 3341, 3344, 3369, 3377, 3381, 3390,
3398, 3408, 3455, 3540, 3586, 3588, 3602,
3608, 3611, 4148, 4149, 4159, 4227, 4237, 4239
\l_@@_initial_j_int .....
..... 3108, 3267, 3342, 3349,
3354, 3360, 3370, 3378, 3382, 3391, 3399,
3409, 3411, 3456, 3478, 3481, 3489, 3676,
3678, 3683, 4152, 4162, 4164, 4223, 4224, 4235
\l_@@_initial_open_bool .....
..... 3111, 3343, 3347, 3350,
3357, 3363, 3367, 3383, 3537, 3576, 3583,
3593, 3659, 3666, 3672, 3726, 3774, 3915,
3962, 4146, 4153, 4165, 4221, 4228, 4240, 4298
\@@_insert_tabularnotes: ..... 2462, 2465
\@@_instruction_of_type:nnn .....
..... 1081, 4005, 4021, 4037, 4060, 4084
\c_@@_integers_alist_tl ..... 7088, 7099
\l_@@_inter_dots_dim .....
..... 505, 507, 3116, 3920, 3927,
3938, 3946, 3953, 3958, 3970, 3978, 5130, 5140
\g_@@_internal_code_after_tl .....
..... 269, 1873, 2101, 2122,
2130, 2143, 2149, 2155, 2215, 3141, 3142,
5196, 5291, 5313, 5480, 5798, 6141, 6636, 6859
\@@_intersect_our_row:nnnn ..... 4630
\@@_intersect_our_row_p:nnnn ..... 4580
\@@_j: ..... 5545, 5547,
5548, 5549, 5550, 5555, 5558, 5560, 5563,
5565, 5566, 5568, 5571, 5573, 5574, 5589,
5592, 5594, 5595, 5596, 5651, 5653, 5656,
5658, 5662, 5663, 5676, 5679, 5680, 5682,
5687, 5688, 5700, 5706, 5707, 5709, 5714, 5715
\l_@@_key_nb_rows_int .....
..... 230, 4349, 4369, 4377, 4384
\l_@@_l_dim .....
.... 3893, 3894, 3907, 3908, 3920, 3926,
3937, 3945, 3953, 3958, 3970, 3971, 3978, 3979
\l_@@_large_nodes_bool 533, 636, 5530, 5534
\g_@@_last_col_found_bool ..... 341,
1269, 1614, 1682, 2739, 2818, 2897, 3032, 6277
\l_@@_last_col_int .....
.. 339, 340, 801, 841, 843, 856, 872, 893,
1290, 1293, 1617, 1764, 2969, 2971, 3033,
3035, 3646, 3681, 4002, 4018, 4056, 4080,
6027, 6032, 6033, 6034, 6037, 6073, 6083,
6099, 6111, 6123, 6135, 6147, 6162, 6203,
6208, 6216, 6232, 6585, 6591, 6597, 7417, 7440
\l_@@_last_col_without_value_bool ...
..... 338, 840, 3034, 7420
\l_@@_last_empty_column_int .....
..... 5409, 5410, 5423, 5429, 5442
\l_@@_last_empty_row_int .....
..... 5391, 5392, 5405, 5426
\l_@@_last_i_tl .....
.... 6880, 6891, 6892, 6902, 6915, 6949,
6962, 6964, 7012, 7019, 7201, 7218, 7239, 7299
\l_@@_last_j_tl .....
..... 6881, 6894, 6895, 6903, 6926,
6928, 6985, 6995, 7002, 7202, 7242, 7244, 7254
\l_@@_last_row_int .....
335, 336, 599, 922, 968, 1151, 1284, 1288,
1295, 1602, 1606, 1609, 1621, 1643, 2621,
2622, 2861, 2862, 2906, 2907, 3037, 3526,
3565, 4034, 4050, 4074, 4767, 4775, 6026,
6029, 6030, 6049, 6073, 6083, 6099, 6111,
6123, 6134, 6146, 6160, 6231, 6243, 6593, 7783
\l_@@_last_row_without_value_bool ...
..... 337, 1286, 1604, 3036
\l_@@_left_delim_dim .....
..... 1314, 1318, 1323, 2587, 2884
\g_@@_left_delim_tl .....
1322, 1513, 1654, 1678, 1748, 2086, 2088, 5129
\l_@@_left_margin_dim .....
..... 535, 639, 1328, 2885, 5126, 5667
\l_@@_letter_for_dotted_lines_str ...
..... 822, 833, 1814, 7733
\l_@@_letter_str ..... 5210,
5229, 5233, 5235, 5238, 5243, 5268, 7621, 7632
\l_@@_letter_vlism_tl ..... 254, 616, 1816
\l_@@_light_syntax_bool 517, 593, 1331, 1564
\@@_light_syntax_i ..... 2612, 2615
\@@_line ..... 3140, 4282
\@@_line_i:nn ..... 4289, 4296
\l_@@_line_width_dim .....
313, 5988, 6325, 6363, 6374, 6380, 6403,
6426, 6458, 6484, 6486, 6503, 6504, 6506, 6524
\@@_line_with_light_syntax:n ... 2626, 2630
\@@_line_with_light_syntax:i:n .....
..... 2625, 2631, 2632, 2640
\l_@@_local_end_int .....
..... 4817, 4838, 4846, 4896, 4945,
4960, 4998, 5019, 5027, 5077, 5132, 5134, 5155
\l_@@_local_start_int .....
..... 4816, 4832, 4833, 4836,
4840, 4844, 4892, 4943, 4956, 4997, 5013,
5014, 5017, 5021, 5025, 5073, 5122, 5124, 5151
\@@_math_toggle_token: ..... 180, 1002,
2855, 2872, 2901, 2917, 6681, 6692, 7285, 7321
\g_@@_max_cell_width_dim .....
..... 1011, 1012, 1536, 2714, 5493, 5519
\c_@@_max_l_dim ..... 3907, 3912
\l_@@_medium_nodes_bool 532, 635, 5528, 6223
\@@_message_hdotsfor: 7426, 7439, 7446, 7454
\@@_msg_new:nn 18, 7366, 7375, 7431, 7436,
7443, 7451, 7459, 7466, 7471, 7477, 7482,
7487, 7492, 7497, 7502, 7507, 7528, 7534,
7540, 7545, 7551, 7557, 7562, 7569, 7576,
7582, 7591, 7596, 7598, 7603, 7610, 7619,
7624, 7629, 7635, 7641, 7658, 7665, 7670,
7677, 7687, 7693, 7700, 7707, 7712, 7722,
7728, 7736, 7743, 7750, 7756, 7762, 7769,
7775, 7781, 7789, 7791, 7797, 7802, 7819, 8124
\@@_msg_new:nnn .....
. 19, 7331, 7512, 7647, 7808, 7825, 7837,

```

7852, 7873, 7892, 7909, 7953, 8006, 8058, 8110	\@@_old_pgfpoinanchor 194, 7080, 7084
\@@_msg_redirect_name:nn	\@@_old_pgful@check@rerun 101, 105
... 20, 660, 812, 819, 1694, 6040, 6043, 7359	\@@_old_vdots 1232, 4042
\@@_multicolumn:nnn 1259, 2245	\@@_open_x_final_dim:
\g_@@_multicolumn_cells_seq 3494, 3546, 3580, 3734, 3783
..... 299, 1262, 1309, 2260,	\@@_open_x_initial_dim:
3058, 3062, 3063, 5563, 5571, 5693, 6187, 6205 3472, 3539, 3577, 3729, 3777
\g_@@_multicolumn_sizes_seq	\@@_open_y_final_dim: 3618, 3670, 3782
.... 300, 1263, 1310, 2262, 3064, 3065, 5694	\@@_open_y_initial_dim:
\l_@@_multiplicity_int 3600, 3667, 3728, 3776
... 4789, 4900, 4912, 4922, 5081, 5091, 5101	\l_@@_other_keys_tl .. 4809, 4878, 4991, 5059
\g_@@_name_env_str 258, 264,	\@@_overbrace_i:n 7260, 7265
265, 1520, 1521, 2643, 2941, 2942, 2950,	\l_@@_parallelize_diags_bool
2951, 2981, 2994, 3012, 3022, 3160, 6606, 7415 521, 522, 631, 3098, 3738, 3786
\l_@@_name_str	\@@_patch_for_revtext: 1490, 1509
... 531, 694, 940, 943, 1043, 1046, 1122,	\@@_patch_m_preamble:n
1125, 2660, 2661, 2685, 2686, 2703, 2704, 2254, 2289, 2322, 2327, 2388
2734, 2735, 2761, 2764, 2811, 2814, 2833,	\@@_patch_m_preamble_i:n
2837, 3082, 3087, 3093, 3240, 3241, 3252, 2293, 2294, 2295, 2309
3255, 3258, 5684, 5687, 5711, 5714, 6172, 6175	\@@_patch_m_preamble_ii:nn
\g_@@_names_seq 237, 691, 693, 8122 2296, 2297, 2298, 2319
\l_@@_nb_cols_int	\@@_patch_m_preamble_iii:n 2299, 2324
..... 6559, 6570, 6578, 6584, 6590, 6596	\@@_patch_m_preamble_iv:nnn
\l_@@_nb_rows_int 6558, 6569, 6587 2300, 2301, 2302, 2329
\@@_newcolumnntype 1158, 1713, 1714, 1715, 3000	\@@_patch_m_preamble_ix:n 2373, 2391
\@@_node_for_cell:	\@@_patch_m_preamble_v:nnnn 2303, 2304, 2349
..... 1020, 1027, 1397, 2882, 2932	\@@_patch_m_preamble_x:n
\@@_node_for_multicolumn:nn 5695, 5702 2317, 2347, 2368, 2370, 2394
\@@_node_left:nn 7061, 7062, 7122	\@@_patch_node_for_cell: 1053, 1397
\@@_node_position:	\@@_patch_node_for_cell:n 1051, 1077, 1080
1353, 1355, 1359, 1361, 1426, 1428, 1436, 1442	\@@_patch_preamble:n
\@@_node_position_i: 1439, 1443 1736, 1782, 1822, 1828, 1847, 1883,
\@@_node_right:nnnn 7071, 7073, 7146	2090, 2106, 2108, 2124, 2132, 2158, 2217, 2237
\g_@@_not_empty_cell_bool	\@@_patch_preamble_i:n 1786, 1787, 1788, 1833
..... 279, 1018, 1025, 2989	\@@_patch_preamble_ii:nn
\@@_not_in_exterior:nnnnn 4622 1789, 1790, 1791, 1844
\@@_not_in_exterior_p:nnnnn 4558	\@@_patch_preamble_iii:n . 1792, 1849, 1857
\l_@@_notes_above_space_dim 524, 526	\@@_patch_preamble_iii_i:n 1852, 1854
\l_@@_notes_bottomrule_bool	\@@_patch_preamble_iv:n
..... 712, 859, 887, 2488 1793, 1794, 1795, 1905
\l_@@_notes_code_after_tl 710, 2497	\@@_patch_preamble_iv_i:n 1908, 1910
\l_@@_notes_code_before_tl 708, 2469	\@@_patch_preamble_iv_ii:w 1913, 1914, 1916
\l_@@_notes_detect_duplicates_bool ..	\@@_patch_preamble_iv_iii:nn ... 1917, 1918
..... 239, 240, 413, 740	\@@_patch_preamble_iv_iv:nn
\@@_notes_format:n 371, 425, 430 1922, 1924, 2025, 2028, 2187
\@@_notes_label_in_list:n 378, 397, 405, 720	\@@_patch_preamble_iv_v:nnnnnnnn 1928, 1961
\@@_notes_label_in_tabular:n . 377, 436, 717	\@@_patch_preamble_ix:nn
\l_@@_notes_labels_seq 369, 424, 429, 439, 444 1804, 1805, 1806, 2110
\l_@@_notes_para_bool .. 706, 857, 885, 2473	\@@_patch_preamble_ix_i:nnn 2114, 2136
\@@_notes_style:n 374, 376, 379, 397, 405, 714	\@@_patch_preamble_v:n ... 1796, 1797, 2011
\l_@@_nullify_dots_bool	\@@_patch_preamble_v_i:w . 2014, 2015, 2017
.... 527, 634, 4009, 4025, 4041, 4065, 4088	\@@_patch_preamble_v_ii:nn 2018, 2019
\@@_old_CT@arc@ 1527, 3162	\@@_patch_preamble_vi:nnnn 1798, 1799, 2031
\@@_old_cdots 1231, 4026	\@@_patch_preamble_vii:n 1800, 2054
\@@_old_ddots 1233, 4066	\@@_patch_preamble_vii_i:w 2057, 2058, 2060
\@@_old_dotfill 6622, 6625, 6633	\@@_patch_preamble_vii_ii:n 2061, 2062
\@@_old_dotfill: 1250	\@@_patch_preamble_viii:nn
\l_@@_old_iRow_int 270, 1280, 3209 1801, 1802, 1803, 2081
\@@_old_ialign: 1103, 1227, 3136, 6021	\@@_patch_preamble_viii_i:nn
\@@_old_iddots 1234, 4089 2094, 2097, 2099
\l_@@_old_jCol_int 271, 1282, 3210	\@@_patch_preamble_x:n ... 1807, 1808, 2161
\@@_old_ldots 1230, 4010	\@@_patch_preamble_x_i:w . 2164, 2165, 2167
\@@_old_multicolumn 1261, 4104	\@@_patch_preamble_x_ii:n 2168, 2171

<code>\@@_patch_preamble_xi:n</code>	<code>\@@_rectanglecolor</code>
... 1842, 1959, 2052, 2077, 2208, 2219, 2243 1376, 4364, 4481, 4514, 4531, 4741
<code>\@@_patch_preamble_xii:n</code>	<code>\@@_rectanglecolor:nnn</code> ... 4487, 4496, 4499
<code>\@@_patch_preamble_xiii:n</code>	<code>\@@_renew_NC@rewrite@S:</code> ... 205, 207, 1268
<code>\@@_pgf_rect_node:nnn</code>	<code>\@@_renew_dots:</code>
477, 1440, 6225	1165, 1258
<code>\@@_pgf_rect_node:nnnnn</code>	<code>\l_@@_renew_dots_bool</code>
..... 452, 1482, 5678, 5705, 6164, 6219, 7048	632, 1258, 7347
<code>\c_@@_pgfortikzpicture_tl</code>	<code>\@@_renew_matrix:</code>
..... 46, 51, 3191, 4315, 6448	804, 6538, 7349
<code>\@@_pgfpntanchor:n</code>	<code>\l_@@_respect_arraystretch_bool</code>
7076, 7081	528, 650, 5738, 5818, 5829, 5928, 5945, 6012
<code>\@@_pgfpntanchor_i:nn</code>	<code>\l_@@_respect_blocks_bool</code> 4537, 4554, 4577
7084, 7086	<code>\@@_restore_iRow_jCol:</code>
<code>\@@_pgfpntanchor_ii:w</code>	3161, 3207
7087, 7095	<code>\c_@@_revtex_bool</code>
<code>\@@_pgfpntanchor_iii:w</code>	56, 59, 62, 63, 1509
7108, 7110	<code>\l_@@_right_delim_dim</code>
<code>\@@_picture_position:</code> 1315, 1319, 1325, 2590, 2929
..... 1348, 1355, 1361, 1428, 1442, 1443	<code>\g_@@_right_delim_tl</code> .. 1324, 1514, 1672,
<code>\g_@@_pos_of_blocks_seq</code>	1678, 1749, 2089, 2118, 2119, 2140, 2145, 5139
293, 1308, 1466, 1538, 2263, 3050, 3054,	<code>\l_@@_right_margin_dim</code>
3055, 3168, 4556, 4823, 5004, 5456, 6089, 6646 536, 641, 1568, 2930, 3685, 5136, 5670
<code>\g_@@_pos_of_stroken_blocks_seq</code>	<code>\@@_rotate:</code>
..... 295, 1539, 4827, 5008, 6101	1248, 4275
<code>\g_@@_pos_of_xdots_seq</code>	<code>\g_@@_rotate_bool</code>
..... 294, 1540, 3406, 4825, 5006 245, 977, 1005, 1983, 2046, 2363, 2874,
<code>\l_@@_position_int</code>	2919, 4275, 5822, 5866, 5871, 5932, 5949, 6152
4779, 4810,	<code>\@@_rotate_cell_box:</code>
4818, 4894, 4940, 4958, 4999, 5075, 5119, 5153 965, 1005, 2046, 2363, 2874, 2919, 6152
<code>\g_@@_post_action_cell_tl</code>	<code>\l_@@_rounded_corners_dim</code>
..... 900, 1004, 1992, 2197, 4390, 4401	311, 5977, 6124, 6339, 6340, 6375, 6428, 6522
<code>\@@_pre_array:</code>	<code>\@@_roundedrectanglecolor</code>
1278, 1339, 1561	1377, 4490
<code>\@@_pre_array_i:w</code>	<code>\l_@@_row_max_int</code> 302, 3276, 3419, 3436
1335, 1561	<code>\l_@@_row_min_int</code> 301, 3344, 3417, 3434
<code>\@@_pre_array_ii:</code>	<code>\g_@@_row_of_col_done_bool</code>
1181, 1311 278, 1136, 1519, 2669
<code>\@@_pre_code_before:</code>	<code>\g_@@_row_style_tl</code>
1341, 1418 282, 912, 1542, 1975, 4382, 4383,
<code>\c_@@_preamble_first_col_tl</code> 1753, 2847	4385, 4388, 4399, 4410, 4415, 4426, 4427, 5816
<code>\c_@@_preamble_last_col_tl</code> 1765, 2892	<code>\g_@@_row_total_int</code> 228, 1265, 1345,
<code>\g_@@_preamble_tl</code>	1351, 1424, 1620, 2432, 2539, 3037, 3044,
..... 1515, 1716, 1720, 1724, 1730,	3475, 3497, 4190, 5538, 5552, 5579, 5674,
1744, 1753, 1762, 1765, 1774, 1778, 1820,	6049, 6182, 6200, 6741, 6902, 6916, 7201, 7566
1835, 1846, 1859, 1963, 2033, 2066, 2093,	<code>\@@_rowcolor</code>
2121, 2129, 2142, 2148, 2192, 2213, 2226,	1378, 4373, 4463
2233, 2242, 2253, 2255, 2311, 2321, 2326,	<code>\@@_rowcolor_tabular</code>
2331, 2351, 2377, 2384, 2393, 2596, 2623, 5308	1178, 4735
<code>\@@_provide_pgfsyspdfmark:</code> ... 68, 77, 1510	<code>\@@_rowcolors</code>
<code>\@@_put_box_in_flow:</code>	1379, 4615
1680, 2396, 2589	<code>\@@_rowcolors_i:nnnnn</code>
<code>\@@_put_box_in_flow_bis:nn</code> 1677, 2556	4581, 4617
<code>\@@_put_box_in_flow_i:</code>	<code>\l_@@_rowcolors_restart_bool</code> ... 4540, 4569
2402, 2404	<code>\@@_rowlistcolors</code>
<code>\@@_qpoint:n</code>	1380, 4544, 4616
222, 1474, 1476,	<code>\g_@@_rows_seq</code> . 2618, 2620, 2622, 2624, 2626
1478, 1480, 2407, 2409, 2421, 2437, 2504,	<code>\l_@@_rows_tl</code>
2506, 2522, 2533, 2544, 3228, 3230, 3232, 4459, 4507, 4583, 4600, 4648, 4673
3234, 3244, 3246, 3489, 3511, 3540, 3547,	<code>\l_@@_rule_width_dim</code>
3586, 3588, 3602, 3620, 3676, 3678, 4323,	4801,
4326, 4667, 4671, 4687, 4689, 4892, 4894,	4959, 5154, 5211, 5254, 5260, 5279, 5302, 5311
4896, 4940, 4943, 4945, 4956, 4958, 4960,	<code>\l_@@_rules_color_tl</code>
5073, 5075, 5077, 5119, 5122, 5132, 5151, 273, 568, 1559, 1560, 6977, 6978
5153, 5155, 5584, 5594, 6156, 6158, 6160,	<code>\@@_set_CT@arc@:</code> 182, 1560, 4794, 6978
6162, 6196, 6216, 6246, 6347, 6349, 6356,	<code>\@@_set_CT@arc@_i:</code>
6358, 6483, 6485, 6487, 6501, 6505, 6507,	183, 184
6660, 6662, 6665, 6667, 6734, 6736, 6946,	<code>\@@_set_CT@arc@_ii:</code>
6949, 6987, 7004, 7021, 7233, 7254, 7267, 7299	183, 186
<code>\l_@@_radius_dim</code>	<code>\@@_set_CT@drsc@:</code>
511, 513, 2214, 3115, 3551, 3552, 3987, 5475	188, 4796
<code>\l_@@_real_left_delim_dim</code> 2558, 2573, 2588	<code>\@@_set_CT@drsc@_i:</code>
<code>\l_@@_real_right_delim_dim</code> 2559, 2585, 2591	189, 190
<code>\@@_recreate_cell_nodes:</code>	<code>\@@_set_CT@drsc@_ii:</code>
1364, 1422	189, 192
<code>\g_@@_recreate_cell_nodes_bool</code>	<code>\@@_set_final_coords:</code>
..... 530, 1196, 1364, 1389, 1396, 1401	3445, 3470
	<code>\@@_set_final_coords_from_anchor:n</code> ..
	... 3461, 3550, 3581, 3662, 3671, 3737, 3785
	<code>\@@_set_initial_coords:</code>
	3440, 3459
	<code>\@@_set_initial_coords_from_anchor:n</code> ..
	... 3450, 3543, 3578, 3661, 3668, 3731, 3779

<code>\@@_set_size:n</code>	6556, 6571
<code>\l_@@_siunitx_loaded_bool</code> ...	195, 199, 204
<code>\g_@@_size_seq</code>	1288, 1293, 1343, 1344, 1345, 1346, 3040
<code>\l_@@_small_bool</code>	802, 846, 852, 874, 909, 1191, 2083, 2112, 2856, 2902, 3113
<code>\@@_standard_cline</code>	139, 1236
<code>\@@_standard_cline:w</code>	139, 140
<code>\l_@@_standard_cline_bool</code> ..	502, 580, 1235
<code>\c_@@_standard_tl</code>	515, 516, 3812
<code>\l_@@_start_int</code>	4781, 4819, 5000
<code>\g_@@_static_num_of_col_int</code>	306, 1689, 1737, 6037, 7455, 7474, 7715
<code>\l_@@_stop_loop_bool</code>	3270, 3271, 3303, 3316, 3325, 3338, 3339, 3371, 3384, 3393
<code>\@@_store_in_tmpb_tl</code>	6317, 6319
<code>\@@_stroke_block:nnn</code>	6096, 6321
<code>\@@_stroke_borders_block:nnn</code> ...	6108, 6424
<code>\@@_stroke_borders_block_i:</code>	6441, 6446
<code>\@@_stroke_borders_block_ii:</code> ...	6449, 6453
<code>\@@_stroke_horizontal:n</code> ..	6464, 6466, 6499
<code>\@@_stroke_vertical:n</code>	6460, 6462, 6481
<code>\@@_sub_matrix:nnnnnnn</code>	6871, 6897
<code>\@@_sub_matrix_i:nnnn</code>	6938, 6944
<code>\l_@@_submatrix_extra_height_dim</code>	323, 6787, 6972
<code>\l_@@_submatrix_hlines_clist</code>	328, 6799, 6819, 7011, 7013
<code>\l_@@_submatrix_left_xshift_dim</code>	324, 6789, 7024, 7057
<code>\l_@@_submatrix_name_str</code>	6834, 6906, 7046, 7048, 7060, 7062, 7070, 7074
<code>\g_@@_submatrix_names_seq</code>	297, 3144, 6831, 6835, 7656
<code>\l_@@_submatrix_right_xshift_dim</code>	325, 6791, 7033, 7067
<code>\g_@@_submatrix_seq</code> ...	305, 1306, 3421, 6857
<code>\l_@@_submatrix_slim_bool</code>	6797, 6914, 7587
<code>\l_@@_submatrix_vlines_clist</code>	329, 6801, 6821, 6994, 6996
<code>\l_@@_suffix_tl</code>	5606, 5617, 5627, 5630, 5679, 5687, 5688, 5706, 5714, 5715
<code>\l_@@_tabular_width_dim</code>	243, 1097, 1099, 1776, 3023
<code>\l_@@_tabularnote_tl</code>	368, 861, 889, 2461, 2470
<code>\g_@@_tabularnotes_seq</code>	367, 415, 426, 2476, 2482, 2498
<code>\c_@@_tabularx_loaded_bool</code> ...	40, 41, 3011
<code>\@@_test_hline_in_block:nnnnn</code>	5005, 5007, 5317
<code>\@@_test_hline_in_stroken_block:nnnn</code> .	5009, 5339
<code>\@@_test_if_cell_in_a_block:nn</code>	5395, 5413, 5431, 5451
<code>\@@_test_if_cell_in_block:nnnnnnn</code> ...	5457, 5459
<code>\@@_test_if_math_mode:</code>	248, 1524, 2952
<code>\@@_test_in_corner_h:</code>	5010, 5031
<code>\@@_test_in_corner_v:</code>	4829, 4850
<code>\@@_test_vline_in_block:nnnnn</code>	4824, 4826, 5328
<code>\@@_test_vline_in_stroken_block:nnnn</code> .	4828, 5350
<code>\l_@@_the_array_box</code>	1312, 1327, 1586, 1594, 2449, 2450, 2452, 2455
<code>\c_@@_tikz_loaded_bool</code>	45, 50, 1367, 3124, 5969, 7683
<code>\l_@@_tikz_rule_tl</code>	4798, 4882, 4962, 4963, 5063, 5157, 5158
<code>\l_@@_tikz_seq</code>	309, 5970, 6127, 6136
<code>\l_@@_tmpc_dim</code>	287, 1479, 1486, 3233, 3237, 4669, 4670, 4693, 4897, 4908, 4916, 4921, 4927, 4961, 4965, 5078, 5095, 5100, 5106, 5156, 5160, 6161, 6166, 6221, 6350, 6361, 6486, 6491, 6496, 6666, 6669, 6677
<code>\l_@@_tmpc_int</code>	4572, 4573, 4574, 5256, 5257, 5273, 5304, 5305
<code>\l_@@_tmpc_tl</code> ...	4352, 4360, 4365, 4374, 4502, 4504, 4507, 4666, 4685, 6383, 6395, 6406, 6411, 6436, 6466, 6483, 6852, 6854, 6858
<code>\l_@@_tmpd_dim</code>	288, 1481, 1487, 3235, 3238, 4690, 4693, 4909, 4916, 5088, 5095, 6163, 6166, 6199, 6210, 6214, 6217, 6221, 6359, 6362, 6668, 6669, 6687
<code>\l_@@_tmpd_tl</code>	4503, 4505, 4508, 6384, 6388, 6407, 6418, 6437, 6462, 6501, 6853, 6855, 6858
<code>\g_@@_total_X_weight_int</code>	274, 1184, 1573, 1576, 1595, 2184
<code>\l_@@_type_of_col_tl</code>	844, 845, 2982, 2984, 6563, 6564, 6565, 6573, 6578
<code>\g_@@_types_of_matrix_seq</code>	7405, 7406, 7411, 7415
<code>\@@_underbrace_i:n</code>	7259, 7297
<code>\@@_update_for_first_and_last_row:</code> ..	948, 1013, 1297, 2876, 2921
<code>\@@_use_arraybox_with_notes:</code> ...	1634, 2517
<code>\@@_use_arraybox_with_notes_b:</code> .	1631, 2501
<code>\@@_use_arraybox_with_notes_c:</code>	1632, 1663, 2445, 2515, 2554
<code>\l_@@_v_center_bool</code> ..	6014, 6058, 6063, 6238
<code>\c_@@_varwidth_loaded_bool</code> ...	28, 29, 2024
<code>\@@_vdottedline:n</code>	2216, 5485
<code>\@@_vline:n</code>	1875, 4804, 4983, 5314, 5486, 6392
<code>\@@_vline_i:</code>	4811, 4814
<code>\@@_vline_ii:</code>	4839, 4847, 4875
<code>\@@_vline_iii:</code>	4883, 4887
<code>\@@_vline_iv:</code>	4880, 4935
<code>\@@_vline_v:</code>	4884, 4951
<code>\@@_vlines_block:nnn</code>	6070, 6378
<code>\l_@@_vlines_block_bool</code>	320, 5984, 6066, 6087
<code>\l_@@_vlines_clist</code>	327, 610, 622, 627, 663, 1186, 1722, 1728, 1759, 1771, 2224, 2231, 2375, 2382, 2779, 3123, 4981, 4982
<code>\l_@@_vpos_col_str</code>	1897, 1900, 1902, 1907, 1929, 1945, 2021, 2174
<code>\l_@@_vpos_of_block_tl</code> ..	317, 318, 5732, 5734, 5832, 5846, 5857, 5936, 5954, 6005, 6007
<code>\@@_w:</code>	1713, 1798, 2303
<code>\l_@@_weight_int</code>	2170, 2175, 2176, 2179, 2181, 2182, 2184, 2188
<code>\l_@@_width_dim</code>	236, 797, 882, 1586, 1594, 2992, 2993, 3013, 3014
<code>\g_@@_width_first_col_dim</code>	291, 1518, 1625, 2664, 2877, 2878
<code>\g_@@_width_last_col_dim</code>	290, 1517, 1684, 2822, 2922, 2923

<code>\l_@@_width_used_bool</code>	298, 883, 1571	<code>\alpha</code>	376
<code>\l_@@_x_final_dim</code>	285, 3447, 3496, 3505, 3506, 3509, 3512, 3513, 3664, 3680, 3688, 3692, 3696, 3698, 3703, 3705, 3735, 3744, 3752, 3792, 3800, 3842, 3857, 3866, 3900, 3952, 3968, 4327, 4942, 5133, 5136, 5138, 5140, 6913, 6929, 6930, 6936, 7033, 7050, 7067, 7238, 7245, 7246, 7252, 7255, 7271, 7288, 7303, 7318	<code>\anchor</code>	3219, 3220
<code>\l_@@_x_initial_dim</code>	283, 3442, 3474, 3483, 3484, 3487, 3490, 3491, 3664, 3679, 3680, 3687, 3692, 3696, 3698, 3700, 3703, 3705, 3744, 3752, 3792, 3800, 3839, 3856, 3866, 3900, 3952, 3966, 3968, 3986, 3988, 4324, 4941, 5123, 5126, 5128, 5130, 6912, 6922, 6923, 6933, 7024, 7049, 7057, 7217, 7224, 7225, 7231, 7234, 7271, 7288, 7303, 7318	<code>\array</code>	1498
<code>\l_@@_xdots_color_tl</code>	541, 555, 3530, 3569, 3650, 3651, 3719, 3767, 3821, 4194, 4269, 4286	<code>\arraybackslash</code>	1976, 2199, 2337
<code>\l_@@_xdots_down_tl</code>	561, 3828, 3850, 3885	<code>\arraycolor</code>	1381
<code>\l_@@_xdots_line_style_tl</code>	514, 516, 551, 3812, 3821	<code>\arraycolsep</code>	640, 642, 644, 1096, 1194, 1318, 1319, 1662, 1666, 2450, 2790, 2796, 2806, 5128, 5138
<code>\l_@@_xdots_shorten_dim</code>	508, 510, 559, 3117, 3836, 3837, 3926, 3937, 3945	<code>\arrayrulecolor</code>	115
<code>\l_@@_xdots_up_tl</code>	562, 3827, 3849, 3875	<code>\arrayrulewidth</code>	147, 152, 172, 570, 939, 1112, 1115, 1121, 1152, 1657, 1668, 1725, 1731, 1821, 1867, 2227, 2234, 2378, 2385, 2509, 2548, 2676, 2678, 2684, 2694, 2696, 2702, 2725, 2727, 2733, 2751, 2754, 2760, 2788, 2794, 2806, 2809, 4669, 4670, 4672, 4688, 4690, 4907, 4908, 4911, 4924, 4930, 5090, 5103, 5109, 5193, 5256, 5304, 5519, 6325, 6380, 6403, 6426, 6759, 6972, 6976
<code>\l_@@_y_final_dim</code>	286, 3448, 3548, 3552, 3590, 3594, 3596, 3621, 3631, 3632, 3746, 3749, 3794, 3797, 3842, 3857, 3865, 3902, 3957, 3976, 4328, 4946, 5121, 6737, 6759, 6774, 6950, 6965, 6966, 6971, 6989, 7006, 7050, 7058, 7068	<code>\arraystretch</code>	1193, 3604, 3622, 5819, 5929, 5946, 6948, 6951
<code>\l_@@_y_initial_dim</code>	284, 3443, 3541, 3551, 3589, 3590, 3594, 3596, 3603, 3613, 3614, 3746, 3751, 3794, 3799, 3839, 3856, 3865, 3902, 3957, 3974, 3976, 3986, 3989, 4325, 4944, 5120, 6735, 6759, 6774, 6947, 6958, 6959, 6971, 6988, 7005, 7049, 7058, 7068	<code>\AutoNiceMatrix</code>	6615
<code>\l</code>	2609, 2631, 6585, 6591, 6597, 6715, 7333, 7334, 7372, 7381, 7462, 7474, 7479, 7484, 7489, 7494, 7515, 7548, 7553, 7559, 7566, 7572, 7573, 7588, 7593, 7607, 7621, 7632, 7638, 7644, 7650, 7651, 7662, 7674, 7684, 7696, 7703, 7710, 7719, 7725, 7733, 7740, 7747, 7753, 7759, 7790, 7794, 7799, 7805, 7811, 7822, 7828, 7829, 7839, 7840, 7854, 7855, 7871, 7875, 7876, 7895, 7896, 7912, 7913, 7956, 7957, 8009, 8010, 8061, 8062, 8117, 8118	<code>\AutoNiceMatrixWithDelims</code>	6567, 6607, 6619
<code>\{</code>	265, 1803, 2103, 2128, 2959, 6614, 7029, 7643, 7740, 7956, 8061		
<code>\}</code>	265, 1806, 2103, 2113, 2959, 6614, 7038, 7643, 7740, 7956, 8061		
<code>\ </code>	2961, 6613		
<code>\sqcup</code>	7429, 7439, 7446, 7454, 7455, 7473, 7474, 7565, 7566, 7579, 7585, 7589, 7597, 7637, 7649, 7655, 7667, 7714, 7715, 7716, 7724, 7730, 7738, 7745, 7758, 7784, 7785, 7793, 7828		
		B	
		<code>\baselineskip</code>	118, 124, 2004
		<code>\begingroup</code>	2248
		<code>\bfseries</code>	4419, 4422
		<code>\bgroup</code>	1512
		<code>\Block</code>	1247, 7680, 7690, 7745, 7772, 7811
		<code>\BNiceMatrix</code>	6553
		<code>\bNiceMatrix</code>	6550
		<code>\Body</code>	1335
		<code>\boldmath</code>	4419, 4422
		bool commands:	
		<code>\bool_const:Nn</code>	28, 29, 31, 32, 34, 35, 37, 38, 40, 41, 45, 50, 56, 59, 62, 63
		<code>\bool_do_until:Nn</code>	3271, 3339
		<code>\bool_gset_false:N</code>	977, 1024, 1025, 1135, 1269, 1389, 1519, 1543, 1721, 2888, 2935, 5326, 5337, 5348, 5359, 5871
		<code>\bool_gset_true:N</code>	1546, 1882, 2669, 2852, 2897, 2989, 4011, 4027, 4043, 4067, 4090, 4101, 4275, 4822, 5003
		<code>\bool_if:NTF</code>	181, 413, 726, 735, 906, 909, 1005, 1108, 1136, 1187, 1191, 1196, 1257, 1258, 1287, 1292, 1307, 1364, 1367, 1391, 1394, 1396, 1416, 1509, 1511, 1525, 1535, 1571, 1604, 1682, 1687, 1706, 1711, 1739, 1751, 1983, 2046, 2083, 2112, 2363, 2488, 2655, 2672, 2690, 2708, 2721, 2747, 2784, 2818, 2823, 2856, 2874, 2902, 2919, 3011, 3032, 3034, 3036, 3098, 3113, 3124, 3593, 3595, 3738, 3786, 4009, 4025, 4041, 4065, 4088, 4413, 4554, 4577, 5127, 5137, 5216, 5223, 5225, 5252, 5300, 5404, 5422, 5440, 5502, 5512, 5534, 5818, 5822, 5866, 5928, 5932, 5945, 5949, 6058, 6066, 6076, 6152, 6179, 6223, 6250, 6287, 6626, 7215, 7236, 7384, 7394, 7420, 7587
		<code>\bool_if:nTF</code>	204, 382, 409, 1083, 1614, 3426, 4300, 4632, 4972, 4976, 5167, 5171, 5528, 5779, 6086, 6237, 6277, 6738, 6748, 6750, 6763, 6769, 6778, 7683
	A		
<code>\A</code>	6829		
<code>\aboverulesep</code>	2492		
<code>\addtocounter</code>	442		

\bool_lazy_all:nTF	1755, 1767, 2775, 4898, 5079, 5319, 5330, 5341, 5352, 5825
\bool_lazy_and:nnTF	2447, 2711, 2789, 2795, 2803, 2857, 3582, 3848, 4107, 4643, 5228, 6343, 6998, 7015
\bool_lazy_or:nnTF	548, 1017, 1075, 1747, 2430, 2459, 2537, 2905, 3659, 3811, 3906, 4624, 4656, 4660, 4710, 4714, 5396, 5414, 5432, 5754, 5759, 6901, 7200, 7229, 7250
\bool_lazy_or_p:nn	2860
\bool_not_p:n	1758, 1760, 1770, 1772, 2713, 2778, 2780, 7230, 7251
\bool_set:Nn	3663
\c_false_bool	29, 32, 35, 38, 41, 50, 63, 2123, 2131, 2144, 2150, 2156, 4005, 4021, 4037
\g_tmpa_bool	4822, 4830, 4857, 4865, 4870, 5003, 5011, 5038, 5046, 5051, 5326, 5337, 5348, 5359
\g_tmpb_bool	1721, 1751, 1882
\l_tmpb_bool	5214, 5223, 5280, 5401, 5415, 5433, 5455, 5468
\c_true_bool	28, 31, 34, 37, 40, 45, 56, 59, 62, 2102
box commands:	
\box_clear_new:N	1189, 1312
\box_dp:N	933, 953, 990, 1010, 1065, 1217, 1226, 1302, 2342, 2399, 3622, 5901, 6951
\box_gclear_new:N	5808
\box_grotate:Nn	5868
\box_ht:N	934, 955, 961, 973, 996, 1008, 1057, 1219, 1221, 1224, 1300, 1971, 1995, 1997, 2003, 2338, 2398, 3604, 5892, 6948
\box_ht_plus_dp:N	2567, 2580
\box_move_down:nn	1065, 2001
\box_move_up:nn	84, 86, 88, 1057, 2442, 2515, 2554
\box_rotate:Nn	967
\box_set_dp:Nn	989, 1009, 2399
\box_set_ht:Nn	995, 1007, 2398
\box_set_wd:Nn	983, 2449
\box_use:N	445, 974, 1064, 2006
\box_use_drop:N	1015, 1021, 1040, 2048, 2365, 2401, 2442, 2443, 2455, 2883, 5911, 6272, 6309
\box_wd:N	446, 984, 1012, 1019, 1078, 1323, 1325, 1586, 1594, 2450, 2452, 2574, 2586, 2878, 2881, 2923, 2927, 5880, 6633
\l_tmpa_box	434, 445, 446, 1322, 1323, 1324, 1325, 1649, 2398, 2399, 2401, 2442, 2443, 2567, 2580
\l_tmpb_box	2560, 2574, 2575, 2586
C	
\c	67, 1743
\Cdots	1239, 4016, 4019
\cdots	1168, 1231
\cellcolor	1177, 1375
\centering	1940, 5836
char commands:	
\char_set_catcode_space:n	1699
\chessboardcolors	1383
\cline	175, 1236, 1237
clist commands:	
\clist_clear:N	351, 4703
\clist_if_empty:NnTF	4829, 5010, 6104
\clist_if_empty_p:N	2779
\clist_if_in:NnTF	349, 1144, 1728, 2231, 2382, 4982, 5177, 6459, 6461, 6463, 6465, 6502
\clist_map_inline:Nn	352, 4650, 4673, 4704, 5364, 6996, 7013
\clist_map_inline:nn	2977, 4513, 4562, 6529
\clist_new:N	310, 326, 327, 328, 329, 523
\clist_put_right:Nn	361, 4721
\clist_set:Nn	622, 623, 627, 628, 662, 663, 664
\clist_set_eq:NN	4702
\l_tmpa_clist	351, 361, 363, 4702, 4704
\CodeAfter	901, 1251, 2612, 2615, 2851, 2896, 3143, 7842
\CodeBefore	1507
\color	119, 125, 185, 187, 191, 193, 911, 1653, 1671, 3524, 3527, 3530, 3563, 3566, 3569, 3644, 3647, 3651, 3719, 3767, 4188, 4191, 4194, 4263, 4266, 4269, 4286, 4411, 4450, 5815, 5979, 6768, 7131, 7155, 7211
\colorlet	256, 257, 918, 925, 1183, 2866, 2910
\columncolor	1179, 1382, 3152, 4753
\cr	151, 177, 2845, 7285, 7289, 7319, 7321
\crrcr	2649, 7284, 7316
cs commands:	
\cs_generate_variant:Nn	14, 23, 24, 66, 179, 2640, 2976, 3830, 3845, 4442, 4443, 5297, 5524, 5525
\cs_gset:Npn	119, 125, 5517
\cs_gset_eq:NN	77, 1210, 1527, 3162
\cs_if_exist:NnTF	61, 1280, 1282, 1431, 1528, 1531, 1715, 2064, 3209, 3210, 3306, 3319, 3374, 3387, 3477, 3499, 3607, 3625, 4156, 4174, 4231, 4249, 5218, 5504, 5557, 6184, 6202, 6471, 6743, 6918, 6925, 6954, 6961, 7220, 7241
\cs_if_exist_p:N	549, 4901, 5082, 5398, 5417, 5435
\cs_if_free:NnTF	73, 3002, 3519, 3558, 3639, 3714, 3762
\cs_if_free_p:N	4302, 4304
\cs_new_protected:Npx	3189, 4313, 6446
\cs_set:Nn	714, 717, 720
\cs_set:Npn	115, 116, 121, 122, 127, 139, 140, 154, 156, 157, 163, 165, 185, 187, 191, 193, 379, 1160, 1503, 1504, 2249, 2280, 3004, 3265, 3327, 3395, 4198, 4273, 5181, 5182, 5188, 5189, 5286, 5819, 5929, 5946
\cs_set_eq:NN	1256
\cs_set_nopar:Npn	1098, 1193, 1204, 5699, 5700
\cs_set_nopar:Npx	1099
\cs_set_protected:Npn	6604
\cs_set_protected_nopar:Npn	5796, 6139
D	
\Ddots	1241, 4049, 4050, 4055, 4056
\ddots	1170, 1233
\diagbox	1252, 5796, 6139
dim commands:	
\dim_add:Nn	5668
\dim_compare:nNnTF	118, 124, 981, 987, 993, 1776, 2709, 2927, 2992, 3487, 3509, 3698, 4386, 4397, 5581, 5591, 6194, 6214, 6633
\dim_compare_p:n	5828

I

`\ialign` 1103, 1204, 1227, 3136, 6021
`\iddots` 1242, 4073, 4074, 4079, 4080
`\iddots` 79, 1171, 1234
if commands:
 `\if_mode_math:` 250, 4417
`\IfBooleanTF` 1561
`\ifnum` 129, 4382, 5181, 5204
`\ifstandalone` 1531
`\ignorespaces` 2287, 4428
int commands:
 `\int_case:nnTF` 4047, 4053, 4071, 4077
 `\int_compare:nNnTF` 142, 143, 167, 903, 904,
 915, 922, 958, 968, 1149, 1151, 1284, 1290,
 1295, 1573, 1576, 1602, 1606, 1617, 1621,
 1622, 1689, 1994, 2179, 2258, 2285, 2471,
 2510, 2549, 2621, 2650, 2771, 2903, 3281,
 3288, 3292, 3294, 3349, 3356, 3360, 3362,
 3526, 3565, 3646, 3681, 3683, 4190, 4265,
 4369, 4619, 4664, 4682, 4718, 4749, 4766,
 4767, 4774, 4775, 4810, 4832, 4836, 4844,
 5013, 5017, 5025, 5124, 5134, 5461, 5463,
 5465, 5467, 5814, 5816, 5873, 5885, 6060,
 6243, 6275, 6279, 6352, 6354, 6581, 6583,
 6585, 6589, 6591, 6593, 6595, 6597, 6982, 6984
 `\int_compare_p:n`
 3428, 3429, 3430, 3431, 4634, 4636, 6344, 6345
 `\int_do_until:nNnn` 4574
 `\int_gadd:Nn` 2184, 2284
 `\int_gincr:N`
 902, 931, 1534, 1841, 1958, 2051, 2076,
 2207, 2745, 2774, 2898, 3740, 3788, 5499, 5795
 `\int_if_even:nTF` 4522, 7102
 `\int_incr:N` 1851, 4602
 `\int_min:nn` 3228,
 3230, 3232, 3234, 3244, 3246, 3409, 3436, 3437
 `\int_mod:nn` 4590
 `\int_step_inline:nn` ... 1447, 1453, 3070,
 3076, 3084, 3090, 3226, 4518, 4520, 6995, 7012
 `\int_step_inline:nnnn` 5393, 5411, 5426, 5429
 `\int_use:N` 425
 `\c_zero_int` 2084, 4435, 4625
iow commands:
 `\iow_now:Nn` ... 70, 96, 1698, 1699, 1700, 1705
 `\iow_shipout:Nn` 5514, 5515, 5521
`\item` 2476, 2482

K

`\kern` 89
keys commands:
 `\keys_define:nn`
 ... 24, 544, 566, 573, 653, 704, 744, 751,
 795, 836, 850, 867, 880, 1399, 1887, 2169,
 4093, 4331, 4535, 4777, 4787, 5206, 5241,
 5265, 5488, 5718, 5966, 6369, 6468, 6519,
 6561, 6699, 6704, 6785, 6806, 6815, 7182, 7345
 `\l_keys_key_str` 2170, 7333, 7514,
 7531, 7537, 7660, 7679, 7799, 7810, 7827,
 7839, 7854, 7875, 7894, 7911, 7955, 8008, 8060
 `\keys_set:nn`
 576, 578, 592, 826, 829, 835, 1403, 1411,
 1556, 1557, 1921, 2023, 2070, 2178, 2983,
 2995, 3015, 3024, 3165, 3529, 3568, 3649,

3718, 3766, 4193, 4268, 4285, 4359, 4551,
5501, 6057, 6432, 6706, 6710, 6840, 6907, 7210
`\keys_set_known:nn` 4059, 4083,
4878, 5059, 5215, 5771, 6326, 6381, 6404, 6427
`\keys_set_known:nnN`
 1826, 2177, 4809, 4991, 6574
`\l_keys_value_tl` 7571, 7698, 7705, 8112

L

`\ldots` 1238, 4000, 4003
`\ldots` 1167, 1230
`\leaders` 147, 172
`\left` 1654, 2563, 2578, 6769, 7132, 7156
legacy commands:
 `\legacy_if:nTF` 688
`\line` 3140, 7643, 7850
`\linewidth` 2993

M

`\makebox` 2048, 2365
`\mathinner` 81
mode commands:
 `\mode_leave_vertical:` 1523, 2336, 4411
msg commands:
 `\msg_error:nn` 12
 `\msg_error:nnn` 13
 `\msg_error:nnnn` 15, 6039, 6046, 6050
 `\msg_fatal:nn` 16
 `\msg_fatal:nnn` 17
 `\msg_new:nnn` 18
 `\msg_new:nnnn` 19
 `\msg_redirect_name:nnn` 21
`\multicolumn` . 1259, 1261, 4104, 4113, 4119, 4141
`\multispan` 143, 144, 168, 169, 2247
`\myfiledate` 6
`\myfileversion` 7

N

`\newcolumntype` 3011
`\newcounter` 366, 370
`\NewDocumentCommand` ... 384, 407, 834, 1409,
3164, 3997, 4013, 4029, 4045, 4069, 4127,
4204, 4282, 4357, 4463, 4472, 4481, 4490,
4511, 4516, 4529, 4544, 4615, 4724, 4735,
4747, 6314, 6567, 6615, 6847, 6867, 7172, 7177
`\NewDocumentEnvironment` 1506, 2593,
2602, 2938, 2948, 2979, 2990, 3009, 3020, 5497
`\NewExpandableDocumentCommand` 220, 5741
`\newlist` 388, 399
`\NiceArray` 2997, 3017, 3026
`\NiceArrayWithDelims` 2943, 2953
nicematrix commands:
 `\g_nicematrix_code_after_tl`
 267, 697, 2617,
 3145, 3147, 6068, 6078, 6094, 6106, 6718, 6725
 `\g_nicematrix_code_before_tl` 1276, 3149,
 3156, 4362, 4371, 4728, 4739, 4751, 6117, 6129
`\NiceMatrixLastEnv` 220
`\NiceMatrixOptions` 834, 7912, 8117
`\NiceMatrixoptions` 7702
`\noalign` 118, 124, 129,
152, 1131, 1210, 5181, 5288, 5475, 7286, 7320
`\nobreak` 402
`\nointerlineskip` 7286, 7320

\normalbaselines 1190
 \NotEmpty 1253
 \null 2283
 \nulldelimiterspace 2574, 2586, 6754, 6974
 \nullfont 6765, 6772, 7128, 7135, 7152, 7159

O

\omit 142, 2652, 2668, 2744, 2770, 6714, 6715
 \OnlyMainNiceMatrix 1249, 4758
 \OverBrace 3139, 7206, 7828

P

\par 2470, 2478
 \path 6531
 peek commands:
 \peek_meaning:NTF . 183, 189, 432, 1161, 3005
 \peek_meaning_ignore_spaces:NTF 2595, 5184
 \peek_meaning_remove_ignore_spaces:NTF 175
 \peek_remove_spaces:n
 ... 4726, 4737, 5743, 6849, 6869, 7174, 7179
 \pgfdeclareshape 3212
 \pgfextracty 6246
 \pgfgetlastxy 488
 \pgfpathcircle 3985
 \pgfpathlineto 4921, 4927,
 5100, 5106, 6491, 6511, 6669, 6989, 7006, 7040
 \pgfpathmoveto 4920, 4926,
 5099, 5105, 6490, 6510, 6664, 6988, 7005, 7031
 \pgfpathrectanglecorners 4692, 4914, 5093, 6360
 \pgfpointadd 486
 \pgfpointanchor 194, 223, 3452,
 3463, 3480, 3502, 3610, 3628, 5560, 5568,
 6189, 6207, 6227, 6229, 6247, 6284, 6746,
 6921, 6928, 6957, 6964, 7076, 7080, 7223, 7244
 \pgfpointdiff .. 487, 1355, 1361, 1428, 1442, 1443
 \pgfpointlineattime 3855
 \pgfpointorigin 2659, 2832, 3220
 \pgfpointscale 486
 \pgfpointshapeborder 4323, 4326
 \pgfrememberpicturepositiononpagetrue ...
 936, 1031,
 1119, 1465, 2658, 2682, 2700, 2731, 2758,
 2800, 2829, 3198, 3225, 3809, 4322, 4890,
 4938, 4954, 5071, 5117, 5149, 5603, 5613,
 5624, 6154, 6328, 6455, 6659, 6732, 6909, 7213
 \pgfscope 3852, 6676
 \pgfset 455,
 480, 1032, 6261, 6298, 6675, 6753, 6911, 7257
 \pgfsetbaseline 1030
 \pgfsetcornersarced 4691, 6336
 \pgfsetlinewidth .. 4930, 5109, 6363, 6458, 6976
 \pgfsetrectcap 4931, 5110
 \pgfsetroundcap 6672
 \pgfsetstrokecolor 6334
 \pgfsyspdfmark 73, 74
 \pgftransformrotate 3859
 \pgftransformshift 461, 486, 3236, 3853, 6260,
 6282, 6677, 6687, 6755, 7054, 7064, 7268, 7300
 \pgfusepath 3879, 3889, 4453, 4917, 6364
 \pgfusepathqfill 3991, 5096
 \pgfusepathqstroke
 4932, 5111, 6492, 6512, 6673, 6990, 7007, 7041
 \phantom 4010, 4026, 4042, 4066, 4089
 \pNiceMatrix 6541

prg commands:

\prg_do_nothing:
 77, 205, 563, 1210, 3143, 4784, 4785
 \prg_new_conditional:Nnn 4622, 4630
 \prg_replicate:nn 2740,
 2741, 4141, 4922, 5101, 6584, 6587, 6590, 6596
 \prg_return_false: 4627, 4639
 \prg_return_true: 4628, 4638
 \ProcessKeysOptions 7365
 \ProvideDocumentCommand 79
 \ProvidesExplPackage 4

Q

\quad 403
 quark commands:
 \q_stop 139, 140, 156, 157, 160, 161,
 163, 164, 165, 184, 186, 190, 192, 342, 355,
 1392, 1414, 1560, 1736, 1809, 1882, 2116,
 2138, 2254, 2305, 2612, 2615, 4276, 4290,
 4291, 4501, 4506, 4566, 4654, 4655, 4677,
 4678, 4708, 4709, 4794, 4796, 5697, 5704,
 5746, 5747, 5751, 6116, 6319, 6342, 6351,
 6382, 6385, 6405, 6408, 6435, 6438, 6556,
 6571, 6851, 6856, 6882, 6889, 6978, 7087, 7095

R

\raggedleft 1942, 5837, 6691
 \raggedright 1941, 1983, 5838
 \rectanglecolor 1376, 3151
 \refstepcounter 443
 regex commands:
 \regex_const:Nn 67
 \regex_match:nnTF 6829
 \regex_replace_all:NnN 1741
 \renewcommand 209
 \RenewDocumentEnvironment
 6540, 6543, 6546, 6549, 6552
 \RequirePackage 1, 3, 9, 10, 11
 \right 1672, 2570, 2582, 6778, 7140, 7164
 \rotate 1248
 \roundedrectanglecolor 1377, 6119
 \rowcolor 1178, 1378
 \rowcolors 1379
 \rowlistcolors 1380
 \RowStyle 1254, 7895

S

\savedanchor 3214
 \savenotes 1511
 scan commands:
 \scan_stop: 3146
 \scriptstyle
 909, 2856, 2902, 3840, 3841, 3875, 3885
 seq commands:
 \seq_clear:N 444, 1734
 \seq_clear_new:N 4547, 5363
 \seq_count:N 2622, 4438, 4592
 \seq_gclear:N 1262, 1263, 1305,
 1306, 1308, 1537, 1538, 1539, 1540, 2498, 3144
 \seq_gclear_new:N 1309, 1310, 1388, 2618, 2634
 \seq_gpop_left:NN 2624, 2636
 \seq_gput_left:Nn 693, 2260, 2262, 6089
 \seq_gput_right:Nn 426, 1818, 2263,
 3406, 4437, 5906, 5918, 6101, 6646, 6835, 6857

<code>\seq_gset_from_clist:Nn</code>	3040, 3054, 3062, 3064, 7406
<code>\seq_gset_map_x:NNn</code>	3168, 7411
<code>\seq_gset_split:Nnn</code>	23, 2620, 2635
<code>\seq_if_empty:NnTF</code>	2457, 3050, 3058, 5379, 6127
<code>\seq_if_empty_p:N</code>	4644
<code>\seq_if_in:NnTF</code>	691, 4684, 4854, 4860, 4867, 5035, 5041, 5048, 5563, 5571, 6187, 6205, 6831, 7415
<code>\seq_item:Nn</code>	1288, 1293, 1343, 1344, 1345, 1346, 4611, 4613
<code>\seq_map_break:</code>	418
<code>\seq_map_function:NN</code>	2626
<code>\seq_map_indexed_inline:Nn</code>	415, 4433, 4448
<code>\seq_map_inline:Nn</code>	1255, 1466, 2476, 2482, 2638, 3421, 4581, 4823, 4825, 4827, 5004, 5006, 5008, 5456, 6022, 6980
<code>\seq_mapthread_function:NNN</code>	5692
<code>\seq_new:N</code>	237, 255, 272, 292, 293, 294, 295, 296, 297, 299, 300, 305, 309, 367, 369, 7405
<code>\seq_put_left:Nn</code>	5295
<code>\seq_put_right:Nn</code>	424, 429, 5443, 5970
<code>\seq_set_eq:NN</code>	4556
<code>\seq_set_filter:NNn</code>	4557, 4579
<code>\seq_set_from_clist:Nn</code>	5383
<code>\seq_set_split:Nnn</code>	4548
<code>\seq_use:Nn</code>	6136
<code>\seq_use:Nnnn</code>	438, 3055, 3063, 3065, 5384, 7656, 8122
<code>\l_tmpa_seq</code>	4557, 4579
<code>\l_tmpb_seq</code>	4556, 4557, 4579, 4581
<code>\setcounter</code>	373
<code>\setlist</code>	389, 400, 727, 736
siunitx commands:	
<code>\siunitx_cell_begin:w</code>	1946, 2064, 2071
<code>\siunitx_cell_end:</code>	1947, 2074
skip commands:	
<code>\skip_gadd:Nn</code>	2716
<code>\skip_gset:Nn</code>	2707, 2772
<code>\skip_gset_eq:NN</code>	2714, 2715
<code>\skip_horizontal:N</code>	148, 173, 1328, 1329, 1568, 1569, 1624, 1625, 1661, 1662, 1665, 1666, 1684, 1685, 1725, 1731, 1821, 2214, 2227, 2234, 2378, 2385, 2587, 2588, 2590, 2591, 2663, 2664, 2676, 2678, 2694, 2696, 2718, 2725, 2727, 2746, 2751, 2754, 2773, 2783, 2788, 2790, 2794, 2796, 2822, 2884, 2885, 2886, 2889, 2924, 2929, 2930, 2931
<code>\skip_horizontal:n</code>	446, 1078, 1863, 5311
<code>\skip_vertical:N</code>	152, 1112, 1115, 2467, 2492, 5475
<code>\skip_vertical:n</code>	973, 1657, 1668, 5191, 5290, 7286, 7320
<code>\g_tmpa_skip</code>	2707, 2714, 2715, 2716, 2718, 2746, 2772, 2773
<code>\c_zero_skip</code>	1215
<code>\smash</code>	7165, 7166
<code>\space</code>	264, 265, 7679
<code>\stepcounter</code>	423
str commands:	
<code>\c_backslash_str</code>	264
<code>\str_case:nn</code>	1938, 5834, 6253, 6265, 6290, 6302, 7025, 7034
<code>\str_case:nnTF</code>	1629, 1784, 1950, 2291, 2424, 5366, 7099, 7112
<code>\str_case_e:nnTF</code>	1812
<code>\str_clear_new:N</code>	5209, 5210, 6906
<code>\str_const:Nn</code>	5264
<code>\str_count:N</code>	5235
<code>\str_gclear:N</code>	3160
<code>\str_gset:Nn</code>	1521, 2942, 2951, 2981, 2994, 3012, 3022, 6606
<code>\str_if_empty:NnTF</code>	940, 1043, 1122, 1520, 2660, 2685, 2703, 2734, 2761, 2811, 2833, 2941, 2950, 3082, 3240, 3252, 5233, 5250, 5684, 5711, 5766, 6167, 6172, 7046, 7060, 7070
<code>\str_if_empty_p:N</code>	5229, 5230
<code>\str_if_eq:nnTF</code>	104, 263, 1101, 1856, 1882, 1912, 1929, 1932, 1945, 1946, 1947, 2013, 2056, 2086, 2118, 2140, 2163, 2221, 2372, 2607, 2609, 2643, 4611, 6332, 6723, 7204, 7258
<code>\str_if_eq_p:nn</code>	550, 1748, 1749, 4658, 4662, 4712, 4716, 5756, 5761
<code>\str_if_in:NnTF</code>	2412, 2524
<code>\str_new:N</code>	231, 258, 314, 531, 833
<code>\str_range:Nnn</code>	2416, 2528
<code>\str_set:Nn</code>	232, 315, 690, 822, 1837, 1889, 1891, 1893, 1895, 1897, 1900, 1902, 1907, 1920, 1933, 1935, 2021, 2022, 2039, 2173, 2313, 2356, 5268, 5271, 5720, 5722, 5724, 5726, 5728, 5730, 5732, 5734, 5767, 5770, 5822, 5933, 5950, 5990, 5992, 5994, 5996, 5999, 6002, 6005, 6007, 6276, 6280, 6834
<code>\str_set_eq:NN</code>	694, 5768
<code>\l_tmpa_str</code>	690, 691, 693, 694
<code>\strut</code>	2476, 2482
<code>\strutbox</code>	3604, 3622, 6948, 6951
<code>\SubMatrix</code>	1384, 3137, 6860, 6904, 7572, 7579, 7585, 7589, 7649, 7857
sys commands:	
<code>\sys_if_engine_xetex_p:</code>	1075
<code>\sys_if_output_dvi_p:</code>	1075
T	
<code>\tabcolsep</code>	1095, 1661, 1665
<code>\tabskip</code>	1215
<code>\tabularnote</code>	384, 407, 432, 7738, 7758
<code>\tabularnotes</code>	2481
TeX and L ^A T _E X 2 _ε commands:	
<code>\@BTnormal</code>	1188
<code>\@addamp</code>	1492, 2249
<code>\@addamp@LaTeX</code>	1492
<code>\@addtopreamble</code>	2256
<code>\@array</code>	1499, 1503
<code>\@array@array</code>	1499
<code>\@arraycr</code>	1496
<code>\@arraycr@array</code>	1496
<code>\@arstrut</code>	2281
<code>\@arstrutbox</code>	933, 934, 973, 1217, 1219, 1221, 1224, 1226, 1971, 1982, 1997, 2003, 2338, 2342
<code>\@classx</code>	1494
<code>\@classx@array</code>	1494
<code>\@currenvir</code>	6723
<code>\@depth</code>	6775, 7137, 7161
<code>\@empty</code>	2256
<code>\@finalstrut</code>	1982

<code>\@firstampfalse</code>	2249	<code>\pgfsys@markposition</code>	
<code>\@gobblethree</code>	74	1060, 1068, 1113, 1200,
<code>\@halignto</code>	1098, 1099	1347, 2656, 2677, 2695, 2726, 2752, 2791, 2825
<code>\@height</code>	132, 147, 172, 6773, 7136, 7160	<code>\pgfutil@check@rerun</code>	101, 102
<code>\@ifclassloaded</code>	55, 58, 7386, 7396	<code>\reserved@a</code>	134
<code>\@ifnextchar</code>	1267, 1503	<code>\rvtx@ifformat@geq</code>	61
<code>\@ifpackageloaded</code>	27,	<code>\set@color</code>	5814, 6151
.....	30, 33, 36, 39, 43, 95, 111, 198, 7389, 7399	<code>\tikz@library@external@loaded</code>	1528
<code>\@mainaux</code>	70,	tex commands:	
.....	96, 1698, 1699, 1700, 1705, 5514, 5515, 5521	<code>\tex_mkern:D</code>	83, 85, 87, 90
<code>\@mkpream</code>	1501, 2255	<code>\tex_the:D</code>	212
<code>\@mkpream@array</code>	1501	<code>\textfont</code>	2438
<code>\@preamble</code>	2282	<code>\textit</code>	376
<code>\@preamerr</code>	2249	<code>\textsuperscript</code>	377, 378
<code>\@sharp</code>	2280	<code>\the</code>	1718, 1736, 2252, 2254, 3003
<code>\@tabarray</code>	1100, 1503	<code>\thetabularnote</code>	379
<code>\@tabular</code>	1500	<code>\tikzexternaldisable</code>	1530
<code>\@tabular@array</code>	1500	<code>\tikzset</code>	1369, 1532, 3126, 4962, 5157
<code>\@tempswafalse</code>	1718, 2252	tl commands:	
<code>\@tempswatrue</code>	1717, 2251	<code>\tl_clear:N</code>	5212, 6324, 7209
<code>\@temptokena</code>	211, 212, 1716, 1736, 2250, 2254	<code>\tl_clear_new:N</code>	4502, 4503, 4549,
<code>\@whilesw</code>	1718, 2252	4585, 6431, 6852, 6853, 6878, 6879, 6880, 6881
<code>\@width</code>	132, 6776, 7138, 7162	<code>\tl_const:Nn</code>	
<code>\@xargarraycr</code>	1497	46, 47, 51, 52, 515, 2847, 2892, 7088
<code>\@xargarraycr@array</code>	1497	<code>\tl_count:n</code>	2419, 2531
<code>\@xarraycr</code>	1495	<code>\tl_gclear:N</code>	
<code>\@xarraycr@array</code>	1495	900, 1542, 1549, 1720, 2253, 3142, 3147, 4452
<code>\@xhline</code>	135	<code>\tl_gclear_new:N</code>	
<code>\array@array</code>	1498	1270, 1271, 1272, 1273, 1274, 1275, 1276, 1541
<code>\bBigg@</code>	1322, 1324	<code>\tl_gput_left:Nn</code>	1083, 1753, 4751
<code>\c@MaxMatrixCols</code>	2970, 7448	<code>\tl_gput_right:Nn</code>	
<code>\c@tabularnote</code>	2460, 2471, 2499	1083, 1578, 1765, 1820, 1859, 1873, 2101,
<code>\col@sep</code>	1095, 1096, 1624,	2122, 2130, 2143, 2149, 2155, 2215, 3038,
.....	1685, 2663, 2716, 2783, 2889, 2924, 3491, 3513	3052, 3060, 3153, 4129, 4206, 4362, 4371,
<code>\CT@arc</code>	115, 116	4383, 4388, 4399, 4410, 4440, 4728, 4739,
<code>\CT@arc@</code>	114, 119,	5196, 5291, 5308, 5313, 5381, 5480, 5798,
.....	133, 146, 171, 185, 187, 1527, 2493, 3162,	6068, 6078, 6094, 6106, 6117, 6129, 6141, 6636
.....	4929, 4947, 5108, 5142, 6333, 6457, 6671, 6979	<code>\tl_gset:Nn</code> ..	260, 1513, 1514, 1515, 1702,
<code>\CT@drs</code>	121, 122	1724, 1730, 2088, 2089, 2119, 2145, 3155, 4438
<code>\CT@drsc@</code>	125,	<code>\tl_if_blank:nTF</code>	4465, 4468,
.....	191, 193, 4901, 4902, 4906, 5082, 5083, 5087	4474, 4477, 4483, 4486, 4492, 4495, 4599, 5745
<code>\CT@everycr</code>	1208	<code>\tl_if_blank_p:n</code>	
<code>\CT@row@color</code>	1210	4657, 4661, 4711, 4715, 4902, 5083, 5755, 5760
<code>\endarray@array</code>	1502	<code>\tl_if_empty:NnTF</code>	
<code>\if@firstamp</code>	2249	1139, 1550, 1559, 1652, 1670, 1827, 2470,
<code>\if@tempswa</code>	1718, 2252	3122, 3123, 3149, 4360, 4408, 4596, 4679,
<code>\insert@column</code>	1493	4680, 4882, 5063, 5220, 5813, 6092, 6114,
<code>\insert@column@array</code>	1493	6330, 6488, 6508, 6767, 6977, 7130, 7154, 7211
<code>\NC@</code>	1160, 3004	<code>\tl_if_empty:nTF</code>	
<code>\NC@do</code>	3003	159, 671, 799, 838, 854, 870, 891,
<code>\NC@find</code>	213	1472, 2604, 2631, 3530, 3569, 3650, 3719,
<code>\NC@find@V</code>	1715	3767, 4194, 4269, 4286, 6826, 7097, 7165, 7428
<code>\NC@list</code>	1718, 2252, 3003	<code>\tl_if_empty_p:N</code>	1759, 1771, 3849, 3850
<code>\NC@rewrite@S</code>	209	<code>\tl_if_empty_p:n</code>	2461, 5786
<code>\new@ifnextchar</code>	1267	<code>\tl_if_eq:NnTF</code>	1141,
<code>\newcol@</code>	1162, 1163, 3006, 3007	1722, 2224, 2375, 2400, 2519, 4981, 5129,
<code>\nicematrix@redefine@check@rerun</code> ..	96, 99	5139, 5176, 6883, 6886, 6890, 6893, 6994, 7011
<code>\pgf@relevantforpicturesizefalse</code> ..	1350,	<code>\tl_if_eq:nnTF</code> ..	417, 683, 808, 2116, 2138, 4434
.....	1464, 3199, 3810, 3982, 4447, 4561, 4891,	<code>\tl_if_eq_p:NN</code>	3812
.....	4939, 4955, 5072, 5118, 5150, 5604, 5614,	<code>\tl_if_exist:NnTF</code>	1544
.....	5625, 6155, 6329, 6456, 6658, 6733, 6910, 7214	<code>\tl_if_in:NnTF</code> ..	4565, 4653, 4676, 4707, 5237
<code>\pgfsys@getposition</code>		<code>\tl_if_in:nnTF</code>	354, 2103, 2113, 2128
.....	1348, 1353, 1359, 1426, 1434, 1437	<code>\tl_if_single_token:nTF</code>	615, 821

<code>\tl_if_single_token_p:n</code>	66	1332, 1333, 1547, 1565, 1566, 2965, 2986, 4451
<code>\tl_map_inline:nn</code>	2605	<code>\use:n</code>
<code>\tl_new:N</code>	22,	1926, 4287, 4758, 4963, 5158, 5844, 5855,
247, 254, 259, 267, 268, 269, 273, 280, 282,		5934, 5952, 6390, 6413, 6495, 6515, 6575, 7083
307, 308, 312, 317, 368, 514, 518, 539, 541, 542		<code>\usepackage</code>
<code>\tl_put_left:Nn</code>	1188, 1397	7391, 7401
<code>\tl_put_right:Nn</code>	673, 1198, 1297, 1337, 1553	<code>\usepgfmodule</code>
<code>\tl_range:nnn</code>	104	2
<code>\tl_set:Nn</code>	4507, 4508, 4567, 4583,	V
4586, 4663, 4665, 4681, 4683, 4717, 4719,		vbox commands:
4818, 4999, 5772, 6353, 6355, 6386, 6387,		<code>\vbox:n</code>
6409, 6410, 6439, 6440, 6884, 6887, 6891, 6894		89, 7312
<code>\tl_set_eq:NN</code>	363, 516,	<code>\vbox_set_top:Nn</code>
4504, 4505, 4666, 6383, 6384, 6406, 6407,		970
6436, 6437, 6854, 6855, 6885, 6888, 6892, 6895		<code>\vbox_to_ht:nn</code>
<code>\tl_set_rescan:Nnn</code>		466, 493, 2566, 2579
2619, 3996, 4126, 4203, 4281, 6866		<code>\vbox_to_zero:n</code>
<code>\tl_to_str:n</code>	7412	972
token commands:		<code>\vbox_top:n</code>
<code>\token_to_str:N</code>		7278
7429, 7542, 7547, 7553, 7571,		<code>\vcenter</code>
7579, 7585, 7589, 7643, 7649, 7667, 7680,		1655, 2564, 6770, 7133, 7157, 7794
7689, 7738, 7745, 7758, 7772, 7794, 7810,		<code>\Vdots</code>
7827, 7828, 7841, 7850, 7856, 7895, 7912, 8117		1240, 4032, 4035
		<code>\vdots</code>
		1169, 1232
		<code>\Vdotsfor</code>
		1246
		<code>\vfill</code>
		469, 495
		<code>\vline</code>
		133
		<code>\VNiceMatrix</code>
		6547
		<code>\vNiceMatrix</code>
		6544
		<code>\vrule</code>
		131, 1971, 2338, 2342
		<code>\vskip</code>
		130
		<code>\vtop</code>
		1110
U		X
<code>\UnderBrace</code>	3138, 7205, 7828	<code>\xglobal</code>
<code>\unskip</code>	2486	918, 925, 1183, 2866, 2910
<code>\upbracefill</code>	7319	
use commands:		Z
<code>\use:N</code>	1086,	<code>\Z</code>
		6829

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 colortbl	13
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	14
6.3	Color tools with the syntax of colortbl	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>	36
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 small	37
13.5	The counters iRow and jCol	37
13.6	The option light-syntax	38
13.7	Color of the delimiters	38
13.8	The environment <code>{NiceArrayWithDelims}</code>	39
13.9	The command <code>\OnlyMainNiceMatrix</code>	39
14	Use of Tikz with nicematrix	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	44

16	Technical remarks	45
16.1	Diagonal lines	45
16.2	The empty cells	45
16.3	The option <code>exterior-arraycolsep</code>	46
16.4	Incompatibilities	46
17	Examples	47
17.1	Utilisation of the key <code>'tikz'</code> of the command <code>\Block</code>	47
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	51
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	237
	Index	245