

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

July 1, 2021

Abstract

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

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

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 as MiKTeX, TeXlive or MacTeX.

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

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

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

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

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

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

*This document corresponds to the version 5.17 of `nicematrix`, at the date of 2021/07/01.

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

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

1 The environments of this package

The package `nicematrix` defines the following new environments.

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

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

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

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

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

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

Important

Before the version 5.0, it was mandatory to use, for technical reasons, the letters `L`, `C` et `R` instead of `l`, `c` et `r` in the preambles of the environments of `nicematrix`. If we want to be able to go on using these letters, `nicematrix` must be loaded with the option `define-L-C-R`.

```
\usepackage[define-L-C-R]{nicematrix}
```

This key will probably be deleted in a future version of `nicematrix`.

2 The vertical space between the rows

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

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

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

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

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

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

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

```

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

```

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

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

$$\begin{array}{cccccc} n & 0 & 1 & 2 & 3 & 4 & 5 \\ u_n & 1 & 2 & 4 & 8 & 16 & 32 \end{array}$$

However, it's also possible to use the tools of `booktabs`⁴: `\toprule`, `\bottomrule`, `\midrule`, etc.

```

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

```

1. an item

$$\begin{array}{cccccc} n & 0 & 1 & 2 & 3 & 4 & 5 \\ u_n & 1 & 2 & 4 & 8 & 16 & 32 \end{array}$$

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}

```

⁴The extension `booktabs` is *not* loaded by `nicematrix`.

```

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

```

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

4 The blocks

4.1 General case

In the environments of `nicematrix`, it's possible to use the command `\Block` in order to place an element in the center of a rectangle of merged cells of the array.⁵ The command `\Block` must be used in the upper leftmost cell of the array with two arguments.

- The first argument is the size of the block with the syntax i - j where i is the number of rows of the block and j its number of columns.
If this argument is empty, its default value is 1-1. If the number of rows is not specified, or equal to *, the block extends until the last row (idem for the columns).
- The second argument is the content of the block. It's possible to use `\\` in that content to have a content on several lines. In `{NiceTabular}` the content of the block is composed in text mode whereas, in the other environments, it is composed in math mode.

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

```

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

```

$$\left[\begin{array}{ccc|c} A & & & 0 \\ & \hspace*{1cm} & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array} \right]$$

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

```

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

```

$$\left[\begin{array}{ccc|c} A & & & 0 \\ & \hspace*{1cm} & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array} \right]$$

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

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

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

```

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

```

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

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

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;
- 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` is in force);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt⁷);
- the key `borders` provides the ability to draw only some borders of the blocks; the value of that key is a (comma-separated) list of elements covered by `left`, `right`, `top` and `bottom`;
- the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`);
- **New 5.15** the keys `hvlines` draws all the vertical and horizontal rules in the block.

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

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

```

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

```

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

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

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.
- 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`.⁸
- It's possible to draw one or several borders of the cell with the key `borders`.

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

We recall that if the first mandatory argument of `\Block` is left blank, the block is mono-cell.⁹

⁸If 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).

⁹One may consider that the default value of the first mandatory argument of `\Block` is 1-1.

4.5 Horizontal position of the content of the block

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

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

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

New 5.17 In order to have an horizontal positioning of the content of the block computed with the limits of the columns of the LaTeX array (and not with the contents of those columns), one may use the key L, R and C of the command `\Block`.

5 The rules

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

5.1 Some differences with the classical environments

5.1.1 The vertical rules

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

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

First	Second
Peter	
Mary	George

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

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

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

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

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

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

However, in this case, it is probably more clever to add a command `\OnlyMainNiceMatrix` (cf. p. 38):

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

5.1.2 The command `\cline`

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

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

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

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

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

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

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

5.2 The thickness and the color of the rules

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

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

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

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

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

If one wishes to define new specifiers for columns in order to draw vertical rules (for example with a specific color or thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 38.

5.3 The tools of nicematrix for the rules

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

- the keys `hlines`, `vlines` and `hvlines`;
- 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`¹⁰ presented p. 4;
 - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `Vdots`, etc. (cf. p. 19).
- The corners are created by the key `corners` explained below (see p. 10).

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

5.3.1 The keys `hlines` and `vlines`

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

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

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

$$\left(\begin{array}{c|c|c|c|c|c} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{array} \right)$$

5.3.2 The key `hvlines`

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

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

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

New 5.17 The key `hvlines-except-corners` is similar to the key `hvlines` but does not draw the rules on the horizontal and vertical borders of the array.

¹⁰And also the command `\multicolumn` also it’s recommended to use instead `\Block` in the environments of `nicematrix`.

5.3.3 The (empty) corners

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

For each of these corners, we will call *empty corner* (or simply *corner*) the reunion of all the empty rectangles starting from the cell actually in the corner of the array.¹¹

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

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

				A	
			A	A	A
				A	
			A	A	A
A	A	A	A	A	A
A	A	A	A	A	A
	A	A	A		
			A		
			A		

When the key `corners` is used, `nicematrix` computes the (empty) corners and these corners will be taken into account by the tools for drawing the rules (the rules won't be drawn in the corners). *Remark:* In the previous versions of `nicematrix`, there was only a key `hvlines-except-corners` (now considered as obsolete).

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

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

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

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

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

¹¹For sake of completeness, we should also say that a cell contained in a block (even an empty cell) is not taken into account for the determination of the corners. That behaviour is natural.

5.4 The command `\diagbox`

The command `\diagbox` (inspired by the package `diagbox`), allows, when it is used in a cell, to slash that cell diagonally downwards.¹²

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

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

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

5.5 Dotted rules

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

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

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

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

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

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

It's possible to change in `nicematrix` the letter used to specify a vertical dotted line with the option `letter-for-dotted-lines` available in `\NiceMatrixOptions`. Thus released, the letter “:” can be used otherwise (for example by the package `arydshln`¹³).

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 rule¹⁴. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

6 The color of the rows and columns

6.1 Use of `colortbl`

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

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

However, there is two drawbacks:

¹²The author of this document considers that type of construction as graphically poor.

¹³However, one should remark that the package `arydshln` is not fully compatible with `nicematrix`.

¹⁴In fact, with `array`, this is true only for `\hline` and “|” but not for `\cline`: cf p. 8

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

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

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

The package `nicematrix` provides some tools (independent of `colortbl`) to draw the colored panels first, and, then, the content of the cells and the rules. This strategy is more conform to the “painting model” of the formats PostScript and PDF and is more suitable for the PDF viewers. However, it requires several compilations.¹⁵

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

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

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

New commands are available in that `\CodeBefore`: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors`, `\chessboardcolors` and `arraycolor`.¹⁶

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.

New 5.15 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.

¹⁵If you use Overleaf, Overleaf will do automatically the right number of compilations.

¹⁶Remark 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. 36.

```

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

```

a	b	c
e	f	g
h	i	j

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

```

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

```

a	b	c
e	f	g
h	i	j

- The command `\arraycolor` takes in as mandatory argument a color and color the whole tabular with that color (excepted the potential exterior rows and columns: cf. p. 17). It's only a particular case of `\rectanglecolor`.
- The command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

```

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

```

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

We have used the key `r` which aligns all the columns rightwards (cf. p. 31).

- 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).

```

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

```

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`¹⁷. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows of the tabular with the tow colors (provided in second and third argument), beginning with the row whose number is given in first (mandatory) argument.

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

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

- The key `cols` describes a set of columns. The command `\rowcolors` will color only the cells of these columns. The value is a comma-separated list of intervals of the form *i*-*j* (where *i* or *j* may be replaced by *).
- With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.¹⁸
- With the key `respect-blocks` the “rows” alternately colored may extend over several rows if they have to incorporate blocks (created with the command `\Block`: cf. p. 4).

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

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

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

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

¹⁷The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

¹⁸Otherwise, the color of a given row relies only upon the parity of its absolute number.

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
  \rowcolors{1}{blue!15}{}
\Body
  & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
0 & 1 & & & & & & \\
1 & 1 & 1 & & & & & \\
2 & 1 & 2 & 1 & & & & \\
3 & 1 & 3 & 3 & 1 & & & \\
4 & 1 & 4 & 6 & 4 & 1 & & \\
5 & 1 & 5 & 10 & 10 & 5 & 1 & \\
6 & 1 & 6 & 15 & 20 & 15 & 6 & 1 \\
\end{NiceTabular}
```

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

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

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

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

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

6.3 Color tools with the syntax of `colortbl`

It's possible to access the preceding tools with a syntax close to the syntax of `colortbl`. For that, one must use the key `colortbl-like` in the current environment.¹⁹

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

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

¹⁹Up to now, this key is *not* available in `\NiceMatrixOptions`.

```

\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 width of the columns

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` and `W` 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 of an array 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.²⁰

```

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

```

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

²⁰The 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).

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

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

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

But it's also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment `\NiceMatrixBlock` with the option `auto-columns-width`²¹. The environment `\NiceMatrixBlock` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

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

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

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

8 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`.

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

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

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

The dotted lines have been drawn with the tools presented p. 19.

²¹At this time, this is the only usage of the environment `\NiceMatrixBlock` but it may have other usages in the future.

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.²²
- 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. 33) 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}}
$\begin{pNiceArray}{cc|cc}[first-row,last-row=5,first-col,last-col,nullify-dots]
    & C_1 & & \Cdots & & C_4 & & \\
L_1 & & a_{11} & & a_{12} & & a_{13} & & a_{14} & & L_1 & \\
\Vdots & & a_{21} & & a_{22} & & a_{23} & & a_{24} & & \Vdots & \\
\hline
    & & a_{31} & & a_{32} & & a_{33} & & a_{34} & & \\
L_4 & & a_{41} & & a_{42} & & a_{43} & & a_{44} & & L_4 & \\
    & & C_1 & & \Cdots & & C_4 & & & & \\
\end{pNiceArray}$
```

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

Remarks

- As shown in the previous example, the horizontal and vertical rules doesn’t extend in the exterior rows and columns.
- However, if one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 38.

²²The users wishing exteriors columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 25).

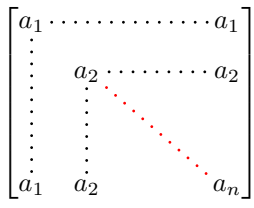
- A specification of color present in `code-for-first-row` also applies to a dotted line draw in this exterior “first row” (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 16) doesn’t apply 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. 25.

9 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`.²³

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells²⁴ 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`.²⁵

```
\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}
```



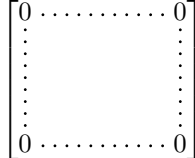
In order to represent the null matrix, one can use the following codage:

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



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}
```



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

²³The 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`.

²⁴The precise definition of a “non-empty cell” is given below (cf. p. 39).

²⁵It’s also possible to change the color of all theses dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 22.

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 & \\
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.²⁶

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

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

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

9.1 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}
```

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

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}
```

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

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

²⁶In `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. 16

9.2 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`²⁷ 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}
```

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

²⁷We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

9.3 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`.²⁸

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`²³ 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}
```

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

9.4 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. 24) 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 \\
& \Ddots^{n \text{ times}} & & \\
0 & & & 1
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & & & 0 \\ \vdots & \ddots & & \\ 0 & & & 1 \end{bmatrix}$$

9.5 Customisation of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and by the command `\line` in the `\CodeAfter` which is described p. 24) may be customized by three options (specified between square brackets after the command):

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

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

²⁸The 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.

- `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. 17.

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).²⁹

```
\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`:

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

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

9.6 The dotted lines and the rules

The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier `l` in the preamble, by the command `\Hline` and by the keys `hlines`, `vlines` and `hvlines` are not drawn within the blocks).³⁰

²⁹The 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.

³⁰On the other side, the command `\line` in the `\CodeAfter` (cf. p. 24) does *not* create block.

```

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

```

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

10 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.³¹

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

The experienced users may, for instance, use the PGF/Tikz nodes created by `nicematrix` in the `\CodeAfter`. These nodes are described further beginning on p. 34.

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

10.1 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. 22).

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

```

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

```

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

It can also be used to draw a diagonal line not parallel to the other diagonal lines (by default, the dotted lines drawn by `\Ddots` are “parallelized”: cf. p. 38).

```

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

```

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

³¹There is also a key `code-before` described p. 12.

10.2 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.³²

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.

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

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

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

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

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

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

³²There 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}]`).

```


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


```

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

```


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


```

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. 37.

New 5.15 It's also possible to specify some delimiters³³ by placing them in the preamble of the environment (for the environments with a preamble: `\NiceArray`, `\pNiceArray`, etc.). This syntax is inspired by the extension `blkarray`.

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

```


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


```

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

11 The notes in the tabulars

11.1 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.

³³Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

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 preferentially. The package `footnote` has some drawbacks, in particular: it must be loaded after the package `xcolor` and it is not perfectly compatible with `hyperref`.

11.2 The notes of tabular

The package `nicematrix` also provides a command `\tabularnote` which gives the ability to specify notes that will be composed at the end of the array with a width of line equal to the width of the array (excepted the potential exterior columns). With no surprise, that command is available only in the environments without delimiters, that is to say `{NiceTabular}`, `{NiceArray}` and `{NiceMatrix}`. In fact, this command is available only if the extension `enumitem` has been loaded (before or after `nicematrix`). Indeed, the notes are composed at the end of the array with a type of list provided by the package `enumitem`.

```
\begin{NiceTabular}{@{}llr@{}}[first-row,code-for-first-row = \bfseries]
\toprule
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. 28. 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.

11.3 Customisation of the tabular notes

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

- `notes/para`
- `notes/bottomrule`
- `notes/style`
- `notes/label-in-tabular`

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

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

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

We detail these keys.

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

Initial value: `false`

That key is also available within a given environment.

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

Initial value: `false`

That key is also available within a given environment.

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

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

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

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

Initial value: `#1`

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

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

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

Initial value: `#1`

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

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

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

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

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

This initial value contains the specification `align = left` which requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 28).

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

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

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

Initial value: `empty`

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

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

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

For an example of customisation of the tabular notes, see p. 40.

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

If you wish to use the environment `{NiceTabular}` or `{NiceTabular*}` 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*} }
\makeatother
```

12 Other features

12.1 Use of the column type S of siunitx

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

```


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


```

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

12.2 Alignment option in `{NiceMatrix}`

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

```


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


```

12.3 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`.

```

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

```

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

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

```

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

```

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

12.4 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`).

```

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

```

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

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

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

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

12.5 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 column³⁴. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 12) and in the `\CodeAfter` (cf. p. 24), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

```

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

```

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

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

³⁴We recall that the exterior “first row” (if it exists) has the number 0 and that the exterior “first column” (if it exists) has also the number 0.

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

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

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

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

12.6 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.

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

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

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

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.³⁵

12.7 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`.

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

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

12.8 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 asymetrical delimiters.

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

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

³⁵The 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 Use of Tikz with nicematrix

13.1 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`. ³⁶

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`.

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. 46).

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

13.2 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`. ³⁷

³⁶One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 19) and the computation of the “corners” (cf. p. 10).

³⁷There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

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

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

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

The names of the “large nodes” are constructed by adding the suffix “`-large`” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.³⁸

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

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`.³⁹

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

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 \\[1ex]
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

³⁸There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 17).

³⁹The 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. 12).

New 5.16 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).

13.3 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`.

	1.5	tulipe	lys
2		2.5	violette mauve
3			3.5
4			

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

```
\begin{NiceMatrix}
\CodeBefore
\tikz \draw [fill=red!15] (7-14) |- (8-15) |- (9-16) |- cycle ;
\Body
1 \\\
1 & 1 \\\
1 & 2 & 1 \\\
1 & 3 & 3 & 1 \\\
1 & 4 & 6 & 4 & 1 \\\
1 & 5 & 10 & 10 & 5 & 1 \\\
1 & 6 & 15 & 20 & 15 & 6 & 1 \\\
1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \\\
1 & 8 & 28 & 56 & 70 & 56 & 28 & 8 & 1
\end{NiceMatrix}
```

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

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

```

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

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

13.4 The nodes corresponding to the command `\SubMatrix`

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 25.

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

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

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

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

14 API for the developpers

The package `nicematrix` provides two variables which are internal but public⁴⁰:

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

These variables contain the code of what we have called the “code-before” and the “code-after”. 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 `\hatchcell` to hatch the current cell (with an optional argument between brackets for the color). It’s possible to program such command `\hatchcell` as follows, explicitly using the public variable `\g_nicematrix_code_before_tl` (this code requires the Tikz library `patterns`: `\usetikzlibrary{patterns}`).

```

ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_hatch:nnn
{
\tikz \fill [ pattern = north-west-lines , pattern-color = #3 ]
( #1 -| #2 ) rectangle ( \int_eval:n { #1 + 1 } -| \int_eval:n { #2 + 1 } ) ;
}

\NewDocumentCommand \hatchcell { ! 0 { black } }
{
\tl_gput_right:Nx \g_nicematrix_code_before_tl
```

⁴⁰According 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.

```

    { \_pantigny_hatch:nnn { \arabic { iRow } } { \arabic { jCol } } { #1 } }
  }
\ExplSyntaxOff

```

Here is an example of use:

```

\begin{NiceTabular}{ccc}[hvlines]
Tokyo & Paris & London \\
Lima & \hatchcell[blue!30]Oslo & Miami \\
Los Angeles & Madrid & Roma
\end{NiceTabular}

```

Tokyo	Paris	London
Lima	Oslo	Miami
Los Angeles	Madrid	Roma

15 Technical remarks

15.1 Definition of new column types

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

For example, one may wish to define a new column type `?` in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job⁴¹:

```

\newcolumntype{?}{!\OnlyMainNiceMatrix{\vrule width 1 pt}}

```

The heavy vertical rule won't extend in the exterior rows.⁴²

```

$\begin{pNiceArray}{cc?cc}[first-row,last-row=3]
C_1 & C_2 & C_3 & C_4 \\
a & b & c & d \\
e & f & g & h \\
C_1 & C_2 & C_3 & C_4
\end{pNiceArray}$

```

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

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

15.2 Diagonal lines

By default, all the diagonal lines⁴³ 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):

```

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

```

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

⁴¹The command `\vrule` is a TeX (and not LaTeX) command.

⁴²Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

⁴³We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

```

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

```

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

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

The same example without parallelization:

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

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

15.3 The “empty” cells

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

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

```

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

```

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

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

15.4 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea⁴⁴. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces

⁴⁴In 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).*

with explicit instructions `\hskip -\arraycolsep`⁴⁵. The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` of `array` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

15.5 Incompatibilities

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

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

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

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

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

16 Examples

16.1 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 11 p. 26.

Let's consider that we wish to number the notes of a tabular with stars.⁴⁶

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument⁴⁷

```
\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}`.

⁴⁵And 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.

⁴⁶Of course, it's realistic only when there is very few notes in the tabular.

⁴⁷In fact: the value of its argument.


```

\NiceMatrixOptions
{
  notes =
  {
    style = \stars{#1} ,
    enumitem-keys =
    {
      widest* = \value{tabularnote} ,
      align = right
    }
  }
}

\begin{NiceTabular}{{}llr{}}[first-row,code-for-first-row = \bfseries]
\toprule
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.

16.2 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}

```

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

An example for a linear system:

```

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

```

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

16.3 Dotted lines which are no longer dotted

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

```

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

```

$$\left(\begin{array}{ccc} 1 & \cdots & 1 \\ \cdots & 0 & 1 \\ & \cdots & \cdots & 1 \\ & 1 & \cdots & 0 \\ & & \cdots & \cdots & 1 \end{array} \right) \begin{array}{l} \\ \leftarrow i \\ \\ \leftarrow j \\ \end{array}$$

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

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

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

16.4 Stacks of matrices

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

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

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

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

```

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

```

```

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

```

```

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

```

```

\end{NiceMatrixBlock}

```

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

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

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

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

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

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

```

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

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

```

`\end{pNiceArray}$`

...

`\end{NiceMatrixBlock}`

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

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

```

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

```

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

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

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

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

16.5 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 block⁴⁸).

```

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

```

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

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don't spread the cells of the array. We recall that, on the other side, the commands `\hline` and `\Hline`, the specifier “|” and the options `hlines`, `vlines` and `hvlines` spread the cells.⁴⁹

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).

```

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

```

⁴⁸We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

⁴⁹For the command `\cline`, see the remark p. 8.

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

However, it's not possible to do a fine tuning. That's why we describe now a method to highlight a row of the matrix. We create a rectangular Tikz node which encompasses the nodes of the second row with the Tikz library `fit`. This Tikz node is filled after the construction of the matrix. In order to see the text *under* this node, we have to use transparency with the `blend mode` equal to `multiply`.

Caution : Some PDF readers are not able to show transparency.⁵⁰

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

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

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

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

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

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

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

```
\[\begin{pNiceArray}{ccc}[last-col]
\CodeBefore [create-cell-nodes]
  \begin{tikzpicture}
    \node [highlight = (1-1) (1-3)] {} ;
    \node [highlight = (2-1) (2-3)] {} ;
    \node [highlight = (3-1) (3-3)] {} ;
  \end{tikzpicture}
\Body
```

⁵⁰In Overleaf, the “built-in” PDF viewer does not show transparency. You can switch to the “native” viewer in that case.

```

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

```

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

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

```

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

```

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

16.6 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`.

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

```

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

```



```

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

```

17 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 `expl3`:

```

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

```

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

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

```

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

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

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

```

Technical definitions

```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

85 \AtBeginDocument

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Idem for `\CT@drs@`.

```

113     \cs_set_protected:Npn \CT@drsc@ { }
114     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
115     \cs_set:Npn \CT@drs@ #1 #2
116     {
117       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
118       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
119     }
120     \cs_set:Npn \hline
121     {
122       \noalign { \ifnum 0 = ` } \fi
123       \cs_set_eq:NN \hskip \vskip
124       \cs_set_eq:NN \vrule \hrule
125       \cs_set_eq:NN \@width \@height
126       { \CT@arc@ \vline }
127       \futurelet \reserved@a
128       \@xhline
129     }
130   }
131 }

```

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

```

132 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
133 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
134 {
135   \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
136   \int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
137   \multispan { \int_eval:n { #2 - #1 + 1 } }

```

```

138 {
139   \CT@arc@
140   \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`⁵¹

```

141   \skip_horizontal:N \c_zero_dim
142 }

```

Our `\everycr` has been modified. In particular, the creation of the `row` node is in the `\everycr` (maybe we should put it with the incrementation of `\c@iRow`). Since the following `\cr` correspond to a “false row”, we have to nullify `\everycr`.

```

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

```

The following version of `\cline` spreads the array of a quantity equal to `\arrayrulewidth` as does `\hline`. It will be loaded excepted if the key `standard-cline` has been used.

```

147 \cs_set:Npn @@_cline

```

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

```

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

```

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

```

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

```

Now, `#1` is the number of the current column and we have to draw a line from the column `#2` to the column `#3` (both included).

```

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

```

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

```

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

```

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

```

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

```

The following command is a small shortcut.

```

167 \cs_new:Npn @@_math_toggle_token:
168 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

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

```

⁵¹See question 99041 on TeX StackExchange.

```

173 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
174 { \cs_set:Npn \CT@arc@ { \color { #1 } } }

175 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

```

The column S of siunitx

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

```

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

```

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

```

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

```

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

```

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

```

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

```

202       \@@_true_c:
203       < { \siunitx_cell_end: \@@_end_Cell: }
204     }
205     \NC@find
206   }
207 }
208 {
209   {
210     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
211     {
212       \renewcommand*{\NC@rewrite@S}[1] []
213       {
214         \@temptokena \exp_after:wN
215         {
216           \tex_the:D \@temptokena
217           > { \@@_Cell: \c_@@_table_collect_begin_tl S {##1} }
218           \@@_true_c:
219           < { \c_@@_table_print_tl \@@_end_Cell: }
220         }
221         \NC@find
222       }

```

```

223     }
224   }
225 }
226 }

```

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

```

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

```

Parameters

For compatibility with versions prior to 5.0, we provide a load-time option `define_L_C_R`. With this option, it's possible to use the letters `L`, `C` and `R` instead of `l`, `c` and `r` in the preamble of the environments of `nicematrix` as it was mandatory before version 5.0.

```

251 \bool_new:N \c_@@_define_L_C_R_bool
252 \cs_new_protected:Npn \@@_define_L_C_R:
253 {
254   \newcolumntype L l
255   \newcolumntype C c
256   \newcolumntype R r
257 }

```

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.

```

258 \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).

```

259 \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.

```

260 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
261 { \int_use:N \g_@@_env_int }

```

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

```

262 \cs_new_protected:Npn \@@_qpoint:n #1
263 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

264 \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).

```

265 \dim_new:N \l_@@_columns_width_dim

```

The following counters will be used to count the numbers of rows and columns of the array.

```

266 \int_new:N \g_@@_row_total_int
267 \int_new:N \g_@@_col_total_int

```

The following token list will contain the type of the current cell (`l`, `c` or `r`). It will be used by the blocks.

```

268 \tl_new:N \l_@@_cell_type_tl
269 \tl_set:Nn \l_@@_cell_type_tl { 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`.

```

270 \dim_new:N \g_@@_blocks_wd_dim

```

Idem pour the mono-row blocks.

```

271 \dim_new:N \g_@@_blocks_ht_dim
272 \dim_new:N \g_@@_blocks_dp_dim

```

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

```

273 \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.

```

274 \bool_new:N \l_@@_in_env_bool

```

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

```

275 \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.

```

276 \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.

```

277 \dim_new:N \l_@@_tabular_width_dim

```


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

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

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

```
279 \bool_new:N \g_@@_rotate_bool
```

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

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

```
281 \cs_new_protected:Npn \@@_test_if_math_mode:
282 {
283   \if_mode_math: \else:
284     \@@_fatal:n { Outside-math-mode }
285   \fi:
286 }
```

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

```
287 \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 analysis of the preamble of the array.

```
288 \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”.

```
289 \colorlet { nicematrix-last-col } { . }
290 \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`).

```
291 \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*.

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

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

```
293 \cs_new:Npn \@@_full_name_env:
294 {
295   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
296     { command \space \c_backslash_str \g_@@_name_env_str }
297     { environment \space \{ \g_@@_name_env_str \} }
298 }
```

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`).

```
299 \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.

```
300 \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.

```
301 \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.

```
302 \int_new:N \l_@@_old_iRow_int
303 \int_new:N \l_@@_old_jCol_int
```

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

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

```
304 \tl_new:N \l_@@_rules_color_tl
```

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.

```
305 \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).

```
306 \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).

```
307 \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`).

```
308 \tl_new:N \l_@@_code_before_tl
309 \bool_new:N \l_@@_code_before_bool
```

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

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

`expl3` provides scratch dimensions `\l_tmpa_dim` and `\l_tmpb_dim`. We creates two more in the same spirit (if they don't exist yet: that's why we use `\dim_zero_new:N`).

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

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

```
316 \bool_new:N \g_@@_empty_cell_bool
```

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

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

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces:

`{imin}{jmin}{imax}{jmax}{options}{contents}`.

The variable is global because it will be modified in the cells of the array.

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

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

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

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

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

```
321 \seq_new:N \g_@@_pos_of_xdots_seq
```

The sequence `\g_@@_pos_of_xdots_seq` will be used when we will draw the rules required by the key `hvlines` (these rules won’t be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key `draw=red!15` when using the command `\Block`). In that case, the rules specified, for instance, by `hvlines` must not be drawn around the block. That’s why we keep the information of all that stroken blocks in the following sequence.

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

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

```
323 \seq_new:N \l_@@_corners_cells_seq
```

The list of the names of the potential `\SubMatrix` in the `\CodeAfter` of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given `\SubMatrix`).

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

The 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).

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

```
326 \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`).

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

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

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

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

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

```
331 \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).

```
332 \int_new:N \g_@@_static_num_of_col_int
```

The following parameters correspond to the keys `fill`, `draw`, `borders` and `rounded-corners` of the command `\Block`.

```
333 \tl_new:N \l_@@_fill_tl
334 \tl_new:N \l_@@_draw_tl
335 \clist_new:N \l_@@_borders_clist
336 \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`.

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

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

```
338 \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`).

```
339 \tl_new:N \l_@@_hpos_of_block_tl
340 \tl_set:Nn \l_@@_hpos_of_block_tl { c }
341 \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`.

```
342 \tl_new:N \l_@@_vpos_of_block_tl
343 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

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

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

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

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

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

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

Variables for the exterior rows and columns

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

- **First row**

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

```
354 \int_new:N \l_@@_first_row_int
355 \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.

```
356 \int_new:N \l_@@_first_col_int
357 \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).

```
358 \int_new:N \l_@@_last_row_int
359 \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”.⁵²

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

Idem for `\l_@@_last_col_without_value_bool`

```
361 \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.

```
362 \int_new:N \l_@@_last_col_int
363 \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.

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

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

⁵²We 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.

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.

```
365 \newcounter { tabularnote }
```

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

```
366 \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.

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

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

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

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

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

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

```
370 \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`.

```
371 \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`.

```
372 \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`).

```
373 \AtBeginDocument
374 {
375   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
376   {
377     \NewDocumentCommand \tabularnote { m }
378     { \@@_error:n { enumitem-not-loaded } }
379   }
380   {
```

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.

```
381     \newlist { tabularnotes } { enumerate } { 1 }
382     \setlist [ tabularnotes ]
383     {
384       topsep = 0pt ,
385       noitemsep ,
386       leftmargin = * ,
```

```

387         align = left ,
388         labelsep = Opt ,
389         label =
390         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
391     }
392     \newlist { tabularnotes* } { enumerate* } { 1 }
393     \setlist [ tabularnotes* ]
394     {
395         afterlabel = \nobreak ,
396         itemjoin = \quad ,
397         label =
398         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
399     }

```

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.⁵³

```

400     \NewDocumentCommand \tabularnote { m }
401     {
402         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
403         { \@@_error:n { tabularnote~forbidden } }
404         {

```

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

```

405         \int_incr:N \l_@@_number_of_notes_int

```

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

```

406         \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
407         \peek_meaning:NF \tabularnote
408         {

```

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.

```

409         \hbox_set:Nn \l_tmpa_box
410         {

```

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

```

411         \@@_notes_label_in_tabular:n
412         {
413             \stepcounter { tabularnote }
414             \@@_notes_style:n { tabularnote }
415             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
416             {
417                 ,
418                 \stepcounter { tabularnote }
419                 \@@_notes_style:n { tabularnote }
420             }
421         }
422     }

```

⁵³We should try to find a solution to that problem.

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.

```

423         \addtocounter { tabularnote } { -1 }
424         \refstepcounter { tabularnote }
425         \int_zero:N \l_@@_number_of_notes_int
426         \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).

```

427         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
428     }
429 }
430 }
431 }
432 }

```

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.

```

433 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
434 {
435     \begin { pgfscope }
436     \pgfset
437     {
438         outer~sep = \c_zero_dim ,
439         inner~sep = \c_zero_dim ,
440         minimum~size = \c_zero_dim
441     }
442     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
443     \pgfnode
444     { rectangle }
445     { center }
446     {
447         \vbox_to_ht:nn
448         { \dim_abs:n { #5 - #3 } }
449         {
450             \vfill
451             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
452         }
453     }
454     { #1 }
455     { }
456     \end { pgfscope }
457 }

```

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.

```

458 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
459 {
460     \begin { pgfscope }
461     \pgfset
462     {
463         outer~sep = \c_zero_dim ,
464         inner~sep = \c_zero_dim ,
465         minimum~size = \c_zero_dim
466     }
467     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }

```



```

468 \pgfpointdiff { #3 } { #2 }
469 \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
470 \pgfnode
471 { rectangle }
472 { center }
473 {
474   \vbox_to_ht:nn
475   { \dim_abs:n \l_tmpb_dim }
476   { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
477 }
478 { #1 }
479 { }
480 \end { pgfscope }
481 }

```

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.

```

482 \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`.

```

483 \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`).

```

484 \dim_new:N \l_@@_cell_space_top_limit_dim
485 \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.

```

486 \dim_new:N \l_@@_inter_dots_dim
487 \AtBeginDocument { \dim_set:Nn \l_@@_inter_dots_dim { 0.45 em } }

```

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

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

```

488 \dim_new:N \l_@@_xdots_shorten_dim
489 \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 em } }

```

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

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

```

490 \dim_new:N \l_@@_radius_dim
491 \AtBeginDocument { \dim_set:Nn \l_@@_radius_dim { 0.53 pt } }

```

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

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

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

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

```
495 \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).

```
496 \tl_new:N \l_@@_baseline_tl
497 \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`).

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

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

```
499 \bool_new:N \l_@@_parallelize_diags_bool
500 \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.

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

```
502 \dim_new:N \l_@@_notes_above_space_dim
503 \AtBeginDocument { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

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

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

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

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

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

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

```
506 \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.

```
507 \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”.

```
508 \bool_new:N \l_@@_medium_nodes_bool
509 \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).

```
510 \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`).

```
511 \dim_new:N \l_@@_left_margin_dim
512 \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`.

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

```
515 \tl_new:N \l_@@_end_of_row_tl
516 \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 “:”.

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

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

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

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

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

```
520 \keys_define:nn { NiceMatrix / xdots }
521 {
522   line-style .code:n =
523   {
524     \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.

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

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

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

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

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

```
547 \keys_define:nn { NiceMatrix / Global }
548 {
549   delimiters .code:n =
550     \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
551   delimiters .value_required:n = true ,
552   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
553   rules .value_required:n = true ,
554   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
555   standard-cline .default:n = true ,
556   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
557   cell-space-top-limit .value_required:n = true ,
558   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
559   cell-space-bottom-limit .value_required:n = true ,
560   cell-space-limits .meta:n =
561     {
562       cell-space-top-limit = #1 ,
563       cell-space-bottom-limit = #1 ,
564     } ,
565   cell-space-limits .value_required:n = true ,
566   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
567   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
568   light-syntax .default:n = true ,
569   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
570   end-of-row .value_required:n = true ,
571   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
572   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
573   last-row .int_set:N = \l_@@_last_row_int ,
574   last-row .default:n = -1 ,
575   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
576   code-for-first-col .value_required:n = true ,
577   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
578   code-for-last-col .value_required:n = true ,
579   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
580   code-for-first-row .value_required:n = true ,
581   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
582   code-for-last-row .value_required:n = true ,
583   hlines .clist_set:N = \l_@@_hlines_clist ,
584   vlines .clist_set:N = \l_@@_vlines_clist ,
585   hlines .default:n = all ,
586   vlines .default:n = all ,
```

```

587 vlines-in-sub-matrix .code:n =
588 {
589     \tl_if_single_token:nTF { #1 }
590     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
591     { \@@_error:n { One~letter~allowed } }
592 },
593 vlines-in-sub-matrix .value_required:n = true ,
594 hvlines .code:n =
595 {
596     \clist_set:Nn \l_@@_vlines_clist { all }
597     \clist_set:Nn \l_@@_hlines_clist { all }
598 },
599 hvlines-except-borders .code:n =
600 {
601     \clist_set:Nn \l_@@_vlines_clist { all }
602     \clist_set:Nn \l_@@_hlines_clist { all }
603     \bool_set_true:N \l_@@_except_borders_bool
604 },
605 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.

```

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

```

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

```

625 \keys_define:nn { NiceMatrix / Env }
626 {
627

```

The key `hvlines-except-corners` is now deprecated.

```

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

```

```

641         \tl_put_right:Nn \l_@@_code_before_tl { #1 }
642         \bool_set_true:N \l_@@_code_before_bool
643     }
644 } ,

```

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

```

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

```

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

672 \keys_define:nn { NiceMatrix / notes }
673 {
674     para .bool_set:N = \l_@@_notes_para_bool ,
675     para .default:n = true ,
676     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
677     code-before .value_required:n = true ,
678     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
679     code-after .value_required:n = true ,
680     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
681     bottomrule .default:n = true ,
682     style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
683     style .value_required:n = true ,
684     label-in-tabular .code:n =
685         \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
686     label-in-tabular .value_required:n = true ,
687     label-in-list .code:n =
688         \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
689     label-in-list .value_required:n = true ,
690     enumitem-keys .code:n =
691     {
692         \bool_if:NTF \c_@@_in_preamble_bool
693         {
694             \AtBeginDocument
695             {
696                 \bool_if:NT \c_@@_enumitem_loaded_bool
697                 { \setlist* [ tabularnotes ] { #1 } }

```

```

698     }
699   }
700   {
701     \bool_if:NT \c_@@_enumitem_loaded_bool
702     { \setlist* [ tabularnotes ] { #1 } }
703   }
704 },
705 enumitem-keys .value_required:n = true ,
706 enumitem-keys-para .code:n =
707 {
708   \bool_if:NTF \c_@@_in_preamble_bool
709   {
710     \AtBeginDocument
711     {
712       \bool_if:NT \c_@@_enumitem_loaded_bool
713       { \setlist* [ tabularnotes* ] { #1 } }
714     }
715   }
716   {
717     \bool_if:NT \c_@@_enumitem_loaded_bool
718     { \setlist* [ tabularnotes* ] { #1 } }
719   }
720 },
721 enumitem-keys-para .value_required:n = true ,
722 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
723 }
724 \keys_define:nn { NiceMatrix / delimiters }
725 {
726   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
727   max-width .default:n = true ,
728   color .tl_set:N = \l_@@_delimiters_color_tl ,
729   color .value_required:n = true ,
730 }

```

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

```

731 \keys_define:nn { NiceMatrix }
732 {
733   NiceMatrixOptions .inherit:n =
734   { NiceMatrix / Global } ,
735   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
736   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
737   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
738   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
739   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
740   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
741   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
742   NiceMatrix .inherit:n =
743   {
744     NiceMatrix / Global ,
745     NiceMatrix / Env ,
746   } ,
747   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
748   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
749   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
750   NiceTabular .inherit:n =
751   {
752     NiceMatrix / Global ,
753     NiceMatrix / Env
754   } ,
755   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
756   NiceTabular / rules .inherit:n = NiceMatrix / rules ,

```

```

757 NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
758 NiceArray .inherit:n =
759 {
760     NiceMatrix / Global ,
761     NiceMatrix / Env ,
762 } ,
763 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
764 NiceArray / rules .inherit:n = NiceMatrix / rules ,
765 NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
766 pNiceArray .inherit:n =
767 {
768     NiceMatrix / Global ,
769     NiceMatrix / Env ,
770 } ,
771 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
772 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
773 pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
774 }

```

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

```

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

```

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

```

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

```

784     transparent .code:n =
785     {
786         \@@_renew_matrix:
787         \bool_set_true:N \l_@@_renew_dots_bool
788         \@@_error:n { Key~transparent }
789     } ,
790     transparent .value_forbidden:n = true,

```

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

```

791     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.

```

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

```

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

```


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

```

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

813 \str_new:N \l_@@_letter_for_dotted_lines_str
814 \str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str

```

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

```

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

```

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

```

817 \keys_define:nn { NiceMatrix / NiceMatrix }
818 {
819   last-col .code:n = \tl_if_empty:nTF {#1}
820   {
821     \bool_set_true:N \l_@@_last_col_without_value_bool
822     \int_set:Nn \l_@@_last_col_int { -1 }
823   }
824   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
825   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
826   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
827   small .bool_set:N = \l_@@_small_bool ,
828   small .value_forbidden:n = true ,
829   unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
830 }

```

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

```

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

```

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.

```

833   small .bool_set:N = \l_@@_small_bool ,
834   small .value_forbidden:n = true ,
835   last-col .code:n = \tl_if_empty:nF { #1 }
836   { \@@_error:n { last-col~non-empty~for~NiceArray } }
837   \int_zero:N \l_@@_last_col_int ,
838   notes / para .bool_set:N = \l_@@_notes_para_bool ,
839   notes / para .default:n = true ,
840   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
841   notes / bottomrule .default:n = true ,
842   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,

```

```

843     tabularnote .value_required:n = true ,
844     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
845     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
846     unknown .code:n = \@@_error:n { Unknown~option~for~NiceArray }
847 }
848 \keys_define:nn { NiceMatrix / pNiceArray }
849 {
850     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
851     last-col .code:n = \tl_if_empty:nF {#1}
852         { \@@_error:n { last-col~non-empty~for~NiceArray } }
853         \int_zero:N \l_@@_last_col_int ,
854     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
855     small .bool_set:N = \l_@@_small_bool ,
856     small .value_forbidden:n = true ,
857     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
858     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
859     unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
860 }

```

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

```

861 \keys_define:nn { NiceMatrix / NiceTabular }
862 {
863     notes / para .bool_set:N = \l_@@_notes_para_bool ,
864     notes / para .default:n = true ,
865     notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
866     notes / bottomrule .default:n = true ,
867     tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
868     tabularnote .value_required:n = true ,
869     last-col .code:n = \tl_if_empty:nF {#1}
870         { \@@_error:n { last-col~non-empty~for~NiceArray } }
871         \int_zero:N \l_@@_last_col_int ,
872     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
873     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
874     unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular }
875 }

```

Important code used by {NiceArrayWithDelims}

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

```

876 \cs_new_protected:Npn \@@_Cell:
877 {

```

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

```

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

```

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

```

879     \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.

```

880     \int_compare:nNnT \c@jCol = 1
881     { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }

```

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

```

882   \hbox_set:Nw \l_@@_cell_box
883   \bool_if:NF \l_@@_NiceTabular_bool
884   {
885     \c_math_toggle_token
886     \bool_if:NT \l_@@_small_bool \scriptstyle
887   }

```

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.

```

888   \int_compare:nNnTF \c@iRow = 0
889   {
890     \int_compare:nNnT \c@jCol > 0
891     {
892       \l_@@_code_for_first_row_tl
893       \xglobal \colorlet { nicematrix-first-row } { . }
894     }
895   }
896   {
897     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
898     {
899       \l_@@_code_for_last_row_tl
900       \xglobal \colorlet { nicematrix-last-row } { . }
901     }
902   }
903 }

```

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.

```

904 \cs_new_protected:Npn \@@_begin_of_row:
905 {
906   \int_gincr:N \c@iRow
907   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
908   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
909   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
910   \pgfpicture
911   \pgfrememberpicturepositiononpagetrue
912   \pgfcoordinate
913   { \@@_env: - row - \int_use:N \c@iRow - base }
914   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
915   \str_if_empty:NF \l_@@_name_str
916   {
917     \pgfnodealias
918     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
919     { \@@_env: - row - \int_use:N \c@iRow - base }
920   }
921   \endpgfpicture
922 }

```

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.

```

923 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
924 {
925   \int_compare:nNnTF \c@iRow = 0

```

```

926 {
927   \dim_gset:Nn \g_@@_dp_row_zero_dim
928   { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
929   \dim_gset:Nn \g_@@_ht_row_zero_dim
930   { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
931 }
932 {
933   \int_compare:nNnT \c@iRow = 1
934   {
935     \dim_gset:Nn \g_@@_ht_row_one_dim
936     { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
937   }
938 }
939 }
940 \cs_new_protected:Npn \@@_rotate_cell_box:
941 {
942   \box_rotate:Nn \l_@@_cell_box { 90 }
943   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
944   {
945     \vbox_set_top:Nn \l_@@_cell_box
946     {
947       \vbox_to_zero:n { }
948       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
949       \box_use:N \l_@@_cell_box
950     }
951   }
952   \bool_gset_false:N \g_@@_rotate_bool
953 }
954 \cs_new_protected:Npn \@@_adjust_size_box:
955 {
956   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
957   {
958     \box_set_wd:Nn \l_@@_cell_box
959     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
960     \dim_gzero:N \g_@@_blocks_wd_dim
961   }
962   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
963   {
964     \box_set_dp:Nn \l_@@_cell_box
965     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
966     \dim_gzero:N \g_@@_blocks_dp_dim
967   }
968   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
969   {
970     \box_set_ht:Nn \l_@@_cell_box
971     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
972     \dim_gzero:N \g_@@_blocks_ht_dim
973   }
974 }
975 \cs_new_protected:Npn \@@_end_Cell:
976 {
977   \@@_math_toggle_token:
978   \hbox_set_end:
979   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
980   \@@_adjust_size_box:
981   \box_set_ht:Nn \l_@@_cell_box
982   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
983   \box_set_dp:Nn \l_@@_cell_box
984   { \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”).

```

985 \dim_gset:Nn \g_@@_max_cell_width_dim
986 { \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”.

```

987 \@@_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.

```

988 \bool_if:NTF \g_@@_empty_cell_bool
989 { \box_use_drop:N \l_@@_cell_box }
990 {
991   \bool_lazy_or:nnTF
992     \g_@@_not_empty_cell_bool
993     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
994     \@@_node_for_cell:
995     { \box_use_drop:N \l_@@_cell_box }
996 }
997 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
998 \bool_gset_false:N \g_@@_empty_cell_bool
999 \bool_gset_false:N \g_@@_not_empty_cell_bool
1000 }

```

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

```

1001 \cs_new_protected:Npn \@@_node_for_cell:
1002 {
1003   \pgfpicture
1004   \pgfsetbaseline \c_zero_dim
1005   \pgfrememberpicturepositiononpagetrue
1006   \pgfset
1007   {
1008     inner~sep = \c_zero_dim ,
1009     minimum~width = \c_zero_dim
1010   }
1011   \pgfnode
1012   { rectangle }
1013   { base }
1014   { \box_use_drop:N \l_@@_cell_box }
1015   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1016   { }
1017   \str_if_empty:NF \l_@@_name_str
1018   {
1019     \pgfnodealias
1020     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1021     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1022   }
1023   \endpgfpicture
1024 }

```

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.

```

1025 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1026 {
1027   \cs_new_protected:Npn \@@_patch_node_for_cell:
1028   {
1029     \hbox_set:Nn \l_@@_cell_box
1030     {
1031       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1032       \hbox_overlap_left:n
1033       {
1034         \pgfsys@markposition
1035         { \@@_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.

```

1036         #1
1037       }
1038     \box_use:N \l_@@_cell_box
1039     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1040     \hbox_overlap_left:n
1041     {
1042       \pgfsys@markposition
1043       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1044       #1
1045     }
1046   }
1047 }
1048 }

```

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

```

1049 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1050 {
1051   \@@_patch_node_for_cell:n
1052   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1053 }
1054 { \@@_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.

```

1055 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1056 {
1057   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1058   { g_@@_ #2 _ lines _ tl }

```

```

1059 {
1060   \use:c { @@ _ draw _ #2 : nnn }
1061   { \int_use:N \c@iRow }
1062   { \int_use:N \c@jCol }
1063   { \exp_not:n { #3 } }
1064 }
1065 }

```

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

```

1066 \cs_new_protected:Npn \@@_revtex_array:
1067 {
1068   \cs_set_eq:NN \@acoll \@arrayacol
1069   \cs_set_eq:NN \@acolr \@arrayacol
1070   \cs_set_eq:NN \@acol \@arrayacol
1071   \cs_set_nopar:Npn \@halignto { }
1072   \@array@array
1073 }
1074 \cs_new_protected:Npn \@@_array:
1075 {
1076   \bool_if:NTF \c_@@_revtex_bool
1077   \@@_revtex_array:
1078   {
1079     \bool_if:NTF \l_@@_NiceTabular_bool
1080     { \dim_set_eq:NN \col@sep \tabcolsep }
1081     { \dim_set_eq:NN \col@sep \arraycolsep }
1082     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1083     { \cs_set_nopar:Npn \@halignto { } }
1084     { \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`.

```

1085   \@tabarray
1086 }

```

`\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.

```

1087   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1088 }

```

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.

```

1089 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1090 \cs_new_protected:Npn \@@_create_row_node:
1091 {

```

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

```

1092   \hbox
1093   {
1094     \bool_if:NT \l_@@_code_before_bool
1095     {
1096       \vtop
1097       {
1098         \skip_vertical:N 0.5\arrayrulewidth
1099         \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
1100         \skip_vertical:N -0.5\arrayrulewidth
1101       }
1102     }
1103     \pgfpicture
1104     \pgfrememberpicturepositiononpagetrue

```

```

1105     \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
1106     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1107     \str_if_empty:NF \l_@@_name_str
1108     {
1109         \pgfnodealias
1110         { \l_@@_name_str - row - \int_use:N \c@iRow }
1111         { \@@_env: - row - \int_use:N \c@iRow }
1112     }
1113     \endpgfpicture
1114 }
1115 }

```

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

```

1116 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1117 \cs_new_protected:Npn \@@_everycr_i:
1118 {
1119     \int_gzero:N \c@jCol
1120     \bool_gset_false:N \g_@@_after_col_zero_bool
1121     \bool_if:NF \g_@@_row_of_col_done_bool
1122     {
1123         \@@_create_row_node:

```

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

```

1124     \tl_if_empty:NF \l_@@_hlines_clist
1125     {
1126         \tl_if_eq:NnF \l_@@_hlines_clist { all }
1127         {
1128             \exp_args:NNx
1129             \clist_if_in:NnT
1130             \l_@@_hlines_clist
1131             { \@@_succ:n \c@iRow }
1132         }
1133     }

```

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.

```

1134     \int_compare:nNnT \c@iRow > { -1 }
1135     {
1136         \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@`.

```

1137         { \hrule height \arrayrulewidth width \c_zero_dim }
1138     }
1139 }
1140 }
1141 }
1142 }

```

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`).

```

1143 \cs_set_protected:Npn \@@_newcolumntype #1
1144 {
1145     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1146     \peek_meaning:NTF [
1147         { \newcol@ #1 }
1148         { \newcol@ #1 [ 0 ] }
1149     }

```


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

```

1150 \cs_set_protected:Npn \@@_renew_dots:
1151 {
1152   \cs_set_eq:NN \ldots \@@_Ldots
1153   \cs_set_eq:NN \cdots \@@_Cdots
1154   \cs_set_eq:NN \vdots \@@_Vdots
1155   \cs_set_eq:NN \ddots \@@_Ddots
1156   \cs_set_eq:NN \iddots \@@_Iddots
1157   \cs_set_eq:NN \dots \@@_Ldots
1158   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1159 }

```

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

```

1160 \cs_new_protected:Npn \@@_colortbl_like:
1161 {
1162   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1163   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1164   \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1165 }

```

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

```

1166 \cs_new_protected:Npn \@@_pre_array_ii:
1167 {

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` *before* the vertical skip (and thus, at a wrong place). That why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition ⁵⁴.

```

1168   \bool_if:NT \c_@@_booktabs_loaded_bool
1169     { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1170   \box_clear_new:N \l_@@_cell_box
1171   \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}`).

```

1172   \bool_if:NT \l_@@_small_bool
1173     {
1174       \cs_set_nopar:Npn \arraystretch { 0.47 }
1175       \dim_set:Nn \arraycolsep { 1.45 pt }
1176     }

1177   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1178     {
1179       \tl_put_right:Nn \@@_begin_of_row:
1180         {
1181           \pgfsys@markposition
1182           { \@@_env: - row - \int_use:N \c@iRow - base }
1183         }
1184     }

```

⁵⁴cf. `\nicematrix@redefine@check@rerun`

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`.

```

1185 \cs_set_nopar:Npn \ialign
1186 {
1187   \bool_if:NTF \c_@@_colortbl_loaded_bool
1188   {
1189     \CT@everycr
1190     {
1191       \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1192       \@@_everycr:
1193     }
1194   }
1195   { \everycr { \@@_everycr: } }
1196   \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`⁵⁵ 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`.

```

1197 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1198 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1199 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1200 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1201 \dim_gzero_new:N \g_@@_ht_row_one_dim
1202 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1203 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1204 \dim_gzero_new:N \g_@@_ht_last_row_dim
1205 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1206 \dim_gzero_new:N \g_@@_dp_last_row_dim
1207 \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`.

```

1208 \cs_set_eq:NN \ialign \@@_old_ialign:
1209 \halign
1210 }

```

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).

```

1211 \cs_set_eq:NN \@@_old_ldots \ldots
1212 \cs_set_eq:NN \@@_old_cdots \cdots
1213 \cs_set_eq:NN \@@_old_vdots \vdots
1214 \cs_set_eq:NN \@@_old_ddots \ddots
1215 \cs_set_eq:NN \@@_old_iddots \iddots
1216 \bool_if:NTF \l_@@_standard_cline_bool
1217 { \cs_set_eq:NN \cline \@@_standard_cline }
1218 { \cs_set_eq:NN \cline \@@_cline }
1219 \cs_set_eq:NN \Ldots \@@_Ldots
1220 \cs_set_eq:NN \Cdots \@@_Cdots
1221 \cs_set_eq:NN \Vdots \@@_Vdots
1222 \cs_set_eq:NN \Ddots \@@_Ddots
1223 \cs_set_eq:NN \Iddots \@@_Iddots
1224 \cs_set_eq:NN \hdottedline \@@_hdottedline:
1225 \cs_set_eq:NN \Hline \@@_Hline:
1226 \cs_set_eq:NN \Hspace \@@_Hspace:
1227 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:

```

⁵⁵The option `small of nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1228 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1229 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1230 \cs_set_eq:NN \Block \@@_Block:
1231 \cs_set_eq:NN \rotate \@@_rotate:
1232 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1233 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1234 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1235 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1236 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1237 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1238 \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:

```

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).

```

1239 \seq_gclear:N \g_@@_multicolumn_cells_seq
1240 \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).

```

1241 \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`).

```

1242 \int_gzero_new:N \g_@@_row_total_int

```

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

```

1243 \int_gzero_new:N \g_@@_col_total_int
1244 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1245 \@@_renew_NC@rewrite@S:
1246 \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.

```

1247 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1248 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1249 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1250 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1251 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1252 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1253 \tl_gclear_new:N \g_nicematrix_code_before_tl
1254 }

```

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

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

```

1255 \cs_new_protected:Npn \@@_pre_array:
1256 {
1257   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1258   \int_gzero_new:N \c@iRow
1259   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1260   \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).

```

1261 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1262 {
1263   \bool_set_true:N \l_@@_last_row_without_value_bool
1264   \bool_if:NT \g_@@_aux_found_bool
1265     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \c_@@_size_seq 3 } }
1266 }
1267 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1268 {
1269   \bool_if:NT \g_@@_aux_found_bool
1270     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \c_@@_size_seq 6 } }
1271 }

```

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

```

1272 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1273 {
1274   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1275     {
1276       \dim_gset:Nn \g_@@_ht_last_row_dim
1277         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1278       \dim_gset:Nn \g_@@_dp_last_row_dim
1279         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1280     }
1281 }

1282 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1283 \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.

```

1284 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1285 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1286 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1287 \@@_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.

```

1288 \box_clear_new:N \l_@@_the_array_box

```

If the user has loaded `nicematrix` with the option `define-L-C-R`, he will be able to use `L`, `C` and `R` instead of `l`, `c` and `r` in the preambles of the environments of `nicematrix` (it's a compatibility mode since `L`, `C` and `R` were mandatory before version 5.0).

```

1289 \bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:

```

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

```
1290 \@@_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]`).

```
1291 \dim_zero_new:N \l_@@_left_delim_dim
1292 \dim_zero_new:N \l_@@_right_delim_dim
1293 \bool_if:NTF \l_@@_NiceArray_bool
1294 {
1295   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1296   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1297 }
1298 {
```

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

```
1299 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1300 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1301 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1302 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1303 }
```

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).

```
1304 \hbox_set:Nw \l_@@_the_array_box
1305 \skip_horizontal:N \l_@@_left_margin_dim
1306 \skip_horizontal:N \l_@@_extra_left_margin_dim
1307 \c_math_toggle_token
1308 \bool_if:NTF \l_@@_light_syntax_bool
1309 { \use:c { @@-light-syntax } }
1310 { \use:c { @@-normal-syntax } }
1311 }
```

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

```
1312 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1313 {
1314   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1315   \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...

```
1316 \@@_pre_array:
1317 }
```

The `\CodeBefore`

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

```
1318 \cs_new_protected:Npn \@@_pre_code_before:
1319 {
```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `code-before` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```
1320 \int_set:Nn \c@iRow { \seq_item:Nn \c_@@_size_seq 2 }
```

```

1321 \int_set:Nn \c@jCol { \seq_item:Nn \c_@@_size_seq 5 }
1322 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \c_@@_size_seq 3 }
1323 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \c_@@_size_seq 6 }

```

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

```

1324 \pgfsys@markposition { \@@_env: - position }
1325 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1326 \pgfpicture
1327 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1328 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1329 {
1330   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1331   \pgfcoordinate { \@@_env: - row - ##1 }
1332   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1333 }

```

Now, the recreation of the col nodes.

```

1334 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1335 {
1336   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1337   \pgfcoordinate { \@@_env: - col - ##1 }
1338   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1339 }

```

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

```

1340 \@@_create_diag_nodes:

```

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

```

1341 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1342 \endpgfpicture
1343 \bool_if:NT \c_@@_tikz_loaded_bool
1344 {
1345   \tikzset
1346   {
1347     every~picture / .style =
1348     { overlay , name~prefix = \@@_env: - }
1349   }
1350 }
1351 \cs_set_eq:NN \cellcolor \@@_cellcolor
1352 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1353 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1354 \cs_set_eq:NN \rowcolor \@@_rowcolor
1355 \cs_set_eq:NN \rowcolors \@@_rowcolors
1356 \cs_set_eq:NN \arraycolor \@@_arraycolor
1357 \cs_set_eq:NN \columncolor \@@_columncolor
1358 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1359 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1360 }

```

```

1361 \cs_new_protected:Npn \@@_exec_code_before:
1362 {
1363   \seq_gclear_new:N \g_@@_colors_seq
1364   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1365   \group_begin:

```

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

```

1366 \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`.

```
1367 \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.

```
1368 \@@_actually_color:
1369 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1370 \group_end:
1371 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1372   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1373 }
```

```
1374 \keys_define:nn { NiceMatrix / CodeBefore }
1375 {
1376   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1377   create-cell-nodes .default:n = true ,
1378   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1379   sub-matrix .value_required:n = true ,
1380   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1381   delimiters / color .value_required:n = true ,
1382   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1383 }
1384 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1385 {
1386   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1387   \@@_CodeBefore:w
1388 }
```

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.

```
1389 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1390 {
1391   \bool_if:NT \g_@@_aux_found_bool
1392   {
1393     \@@_pre_code_before:
1394     #1
1395   }
1396 }
```

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.

```
1397 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1398 {
1399   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1400   {
1401     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1402     \pgfcoordinate { \@@_env: - row - ##1 - base }
1403     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1404     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1405     {
1406       \cs_if_exist:cT
1407       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
```

```

1408         {
1409             \pgfsys@getposition
1410             { \@@_env: - ##1 - ####1 - NW }
1411             \@@_node_position:
1412             \pgfsys@getposition
1413             { \@@_env: - ##1 - ####1 - SE }
1414             \@@_node_position_i:
1415             \@@_pgf_rect_node:nnn
1416             { \@@_env: - ##1 - ####1 }
1417             { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1418             { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1419         }
1420     }
1421 }
1422 \@@_create_extra_nodes:
1423 }

```

The environment {NiceArrayWithDelims}

```

1424 \NewDocumentEnvironment { NiceArrayWithDelims }
1425 { m m O { } m ! O { } t \CodeBefore }
1426 {
1427     \@@_provide_pgfsyspdfmark:
1428     \bool_if:NT \c_@@_footnote_bool \savenotes

```

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

```

1429     \bgroup

1430     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1431     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1432     \tl_gset:Nn \g_@@_preamble_tl { #4 }

1433     \int_gzero:N \g_@@_block_box_int
1434     \dim_zero:N \g_@@_width_last_col_dim
1435     \dim_zero:N \g_@@_width_first_col_dim
1436     \bool_gset_false:N \g_@@_row_of_col_done_bool
1437     \str_if_empty:NT \g_@@_name_env_str
1438     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }

```

The following line will be deleted when we will consider that only versions of `siunitx` after v3.0 are compatible with `nicematrix`.

```

1439     \@@_adapt_S_column:
1440     \bool_if:NTF \l_@@_NiceTabular_bool
1441         \mode_leave_vertical:
1442         \@@_test_if_math_mode:
1443     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1444     \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁵⁶. 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.

```

1445     \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.

⁵⁶e.g. `\color[rgb]{0.5,0.5,0}`


```

1446 \cs_if_exist:NT \tikz@library@external@loaded
1447 {
1448   \tikzexternaldisable
1449   \cs_if_exist:NT \ifstandalone
1450   { \tikzset { external / optimize = false } }
1451 }

```

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

```

1452 \int_gincr:N \g_@@_env_int
1453 \bool_if:NF \l_@@_block_auto_columns_width_bool
1454 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

The sequence `\g_@@_blocks_seq` will contain the characteristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks. Of course, this is redundant but it's for efficiency.

```

1455 \seq_gclear:N \g_@@_blocks_seq
1456 \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`.

```

1457 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1458 \seq_gclear:N \g_@@_pos_of_xdots_seq
1459 \tl_gclear_new:N \g_@@_code_before_tl

```

We load all the informations written in the aux file during previous compilations corresponding to the current environment.

```

1460 \bool_gset_false:N \g_@@_aux_found_bool
1461 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1462 {
1463   \bool_gset_true:N \g_@@_aux_found_bool
1464   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1465 }

```

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.

```

1466 \tl_gclear:N \g_@@_aux_tl
1467 \tl_if_empty:NF \g_@@_code_before_tl
1468 {
1469   \bool_set_true:N \l_@@_code_before_bool
1470   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1471 }

```

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`.

```

1472 \bool_if:NTF \l_@@_NiceArray_bool
1473 { \keys_set:nn { NiceMatrix / NiceArray } }
1474 { \keys_set:nn { NiceMatrix / pNiceArray } }
1475 { #3 , #5 }

```

```

1476 \tl_if_empty:NF \l_@@_rules_color_tl
1477 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
1478 % \bigskip

```

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

```

1479 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
1480 }
1481 {
1482   \bool_if:NTF \l_@@_light_syntax_bool
1483   { \use:c { end @@-light-syntax } }

```

```

1484     { \use:c { end @@-normal-syntax } }
1485     \c_math_toggle_token
1486     \skip_horizontal:N \l_@@_right_margin_dim
1487     \skip_horizontal:N \l_@@_extra_right_margin_dim
1488     \hbox_set_end:

```

End of the construction of the array (in the box `\l_@@_the_array_box`).

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).

```

1489     \int_compare:nNnT \l_@@_last_row_int > { -2 }
1490     {
1491         \bool_if:NF \l_@@_last_row_without_value_bool
1492         {
1493             \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1494             {
1495                 \@@_error:n { Wrong~last~row }
1496                 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1497             }
1498         }
1499     }

```

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”.⁵⁷

```

1500     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1501     \bool_if:nTF \g_@@_last_col_found_bool
1502     { \int_gdecr:N \c@jCol }
1503     {
1504         \int_compare:nNnT \l_@@_last_col_int > { -1 }
1505         { \@@_error:n { last~col~not~used } }
1506     }

```

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

```

1507     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1508     \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. 111).

```

1509     \int_compare:nNnT \l_@@_first_col_int = 0
1510     {
1511         \skip_horizontal:N \col@sep
1512         \skip_horizontal:N \g_@@_width_first_col_dim
1513     }

```

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.

Remark that, in all cases, `@@_use_arraybox_with_notes_c:` is used.

```

1514     \bool_if:NTF \l_@@_NiceArray_bool
1515     {
1516         \str_case:VnF \l_@@_baseline_tl
1517         {
1518             b \@@_use_arraybox_with_notes_b:
1519             c \@@_use_arraybox_with_notes_c:
1520         }
1521         \@@_use_arraybox_with_notes:
1522     }

```

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).

⁵⁷We remind that the potential “first column” (exterior) has the number 0.

```

1523 {
1524   \int_compare:nNnTF \l_@@_first_row_int = 0
1525   {
1526     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1527     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1528   }
1529   { \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”.⁵⁸

```

1530   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1531   {
1532     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1533     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1534   }
1535   { \dim_zero:N \l_tmpb_dim }
1536   \hbox_set:Nn \l_tmpa_box
1537   {
1538     \c_math_toggle_token
1539     \tl_if_empty:NF \l_@@_delimiters_color_tl
1540     { \color { \l_@@_delimiters_color_tl } }
1541     \exp_after:wN \left \g_@@_left_delim_tl
1542     \vcenter
1543     {

```

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.

```

1544       \skip_vertical:N -\l_tmpa_dim
1545       \hbox
1546       {
1547         \bool_if:NTF \l_@@_NiceTabular_bool
1548         { \skip_horizontal:N -\tabcolsep }
1549         { \skip_horizontal:N -\arraycolsep }
1550         \@@_use_arraybox_with_notes_c:
1551         \bool_if:NTF \l_@@_NiceTabular_bool
1552         { \skip_horizontal:N -\tabcolsep }
1553         { \skip_horizontal:N -\arraycolsep }
1554       }

```

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

```

1555       \skip_vertical:N -\l_tmpb_dim
1556     }

```

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.

```

1557       \tl_if_empty:NF \l_@@_delimiters_color_tl
1558       { \color { \l_@@_delimiters_color_tl } }
1559       \exp_after:wN \right \g_@@_right_delim_tl
1560       \c_math_toggle_token
1561     }

```

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.

```

1562       \bool_if:NTF \l_@@_delimiters_max_width_bool
1563       {
1564         \@@_put_box_in_flow_bis:nn
1565         \g_@@_left_delim_tl \g_@@_right_delim_tl
1566       }
1567       \@@_put_box_in_flow:
1568     }

```

⁵⁸A 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).

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. 112).

```

1569   \bool_if:NT \g_@@_last_col_found_bool
1570   {
1571     \skip_horizontal:N \g_@@_width_last_col_dim
1572     \skip_horizontal:N \col@sep
1573   }
1574   \bool_if:NF \l_@@_Matrix_bool
1575   {
1576     \int_compare:nNt \c@jCol < \g_@@_static_num_of_col_int
1577     { \@@_error:n { columns-not-used } }
1578   }
1579   \group_begin:
1580   \globaldefs = 1
1581   \@@_msg_redirect_name:n { columns-not-used } { error }
1582   \group_end:
1583   \@@_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.

```

1584   \egroup

```

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

```

1585   \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1586   \iow_now:Nx \@mainaux
1587   {
1588     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _tl }
1589     \iow_newline: { \iow_newline: \exp_not:V \g_@@_aux_tl }
1590   }
1591   \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1592   \bool_if:NT \c_@@_footnote_bool \endsavenotes
1593   }

```

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.

```

1594   \cs_new_protected:Npn \@@_construct_preamble:
1595   {

```

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 programmation which is not in the style of `expl3`.

We redefine the column types `w` and `W`. We use `\@@_newcolumntype` instead of `\newcolumntype` because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters `w` and `W`. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```

1596   \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).

```

1597 \bool_if:NF \l_@@_Matrix_bool
1598 {
1599   \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1600   \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by `expl3`).

```

1601 \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1602 \@tempswatrue

```

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

```

1603 \@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`.

```

1604 \int_gzero:N \c@jCol
1605 \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1606 \bool_gset_false:N \g_tmpb_bool
1607 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1608 {
1609   \tl_gset:Nn \g_@@_preamble_tl
1610     { ! { \skip_horizontal:N \arrayrulewidth } }
1611 }
1612 {
1613   \clist_if_in:NnT \l_@@_vlines_clist 1
1614   {
1615     \tl_gset:Nn \g_@@_preamble_tl
1616       { ! { \skip_horizontal:N \arrayrulewidth } }
1617   }
1618 }

```

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`).

```

1619 \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1620 \int_zero:N \l_tmpa_int

```

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

```

1621 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1622 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1623 }

```

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

```

1624 \bool_if:NT \l_@@_colortbl_like_bool
1625 {
1626   \regex_replace_all:NnN
1627     \c_@@_columncolor_regex
1628     { \c { @@_columncolor_preamble } }
1629   \g_@@_preamble_tl
1630 }

```

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

```

1631 \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}`.

```

1632 \bool_lazy_or:nnT
1633 { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1634 { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1635 { \bool_set_false:N \l_@@_NiceArray_bool }

```

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

```

1636 \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”.

```

1637 \int_compare:nNnTF \l_@@_first_col_int = 0
1638 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1639 {
1640   \bool_lazy_all:nT
1641   {
1642     \l_@@_NiceArray_bool
1643     { \bool_not_p:n \l_@@_NiceTabular_bool }
1644     { \tl_if_empty_p:N \l_@@_vlines_clist }
1645     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1646   }
1647   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1648 }
1649 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1650 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1651 {
1652   \bool_lazy_all:nT
1653   {
1654     \l_@@_NiceArray_bool
1655     { \bool_not_p:n \l_@@_NiceTabular_bool }
1656     { \tl_if_empty_p:N \l_@@_vlines_clist }
1657     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1658   }
1659   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1660 }

```

We add a last column to raise a good error message when the user put 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`).

```

1661 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1662 {
1663   \tl_gput_right:Nn \g_@@_preamble_tl
1664   { > { \@@_error_too_much_cols: } 1 }
1665 }
1666 }

```

```

1667 \cs_new_protected:Npn \@@_patch_preamble:n #1
1668 {
1669   \str_case:nnF { #1 }
1670   {
1671     c { \@@_patch_preamble_i:n #1 }
1672     l { \@@_patch_preamble_i:n #1 }
1673     r { \@@_patch_preamble_i:n #1 }
1674     > { \@@_patch_preamble_ii:nn #1 }
1675     ! { \@@_patch_preamble_ii:nn #1 }
1676     @ { \@@_patch_preamble_ii:nn #1 }
1677     | { \@@_patch_preamble_iii:n #1 }
1678     p { \@@_patch_preamble_iv:nnn t #1 }
1679     m { \@@_patch_preamble_iv:nnn c #1 }
1680     b { \@@_patch_preamble_iv:nnn b #1 }
1681     \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
1682     \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }

```

```

1683 \@@_true_c: { \@@_patch_preamble_vi:n #1 }
1684 ( { \@@_patch_preamble_vii:nn #1 }
1685 [ { \@@_patch_preamble_vii:nn #1 }
1686 \{ { \@@_patch_preamble_vii:nn #1 }
1687 ) { \@@_patch_preamble_viii:nn #1 }
1688 ] { \@@_patch_preamble_viii:nn #1 }
1689 \} { \@@_patch_preamble_viii:nn #1 }
1690 C { \@@_error:nn { old~column~type } #1 }
1691 L { \@@_error:nn { old~column~type } #1 }
1692 R { \@@_error:nn { old~column~type } #1 }
1693 \q_stop { }
1694 }
1695 {
1696 \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1697 { \@@_patch_preamble_xi:n #1 }
1698 {
1699 \str_if_eq:VnTF \l_@@_letter_vlism_tl { #1 }
1700 {
1701 \seq_gput_right:Nx \g_@@_cols_vlism_seq
1702 { \int_eval:n { \c@jCol + 1 } }
1703 \tl_gput_right:Nx \g_@@_preamble_tl
1704 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1705 \@@_patch_preamble:n
1706 }
1707 {
1708 \bool_lazy_and:nnTF
1709 { \str_if_eq_p:nn { : } { #1 } }
1710 \c_@@_arydshln_loaded_bool
1711 {
1712 \tl_gput_right:Nn \g_@@_preamble_tl { : }
1713 \@@_patch_preamble:n
1714 }
1715 { \@@_fatal:nn { unknown~column~type } { #1 } }
1716 }
1717 }
1718 }
1719 }

```

For c, l and r

```

1720 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1721 {
1722 \tl_gput_right:Nn \g_@@_preamble_tl
1723 {
1724 > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } }
1725 #1
1726 < \@@_end_Cell:
1727 }

```

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

```

1728 \int_gincr:N \c@jCol
1729 \@@_patch_preamble_x:n
1730 }

```

For >, ! and @

```

1731 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1732 {
1733 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1734 \@@_patch_preamble:n
1735 }

```

For |

```

1736 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1737 {

```

`\l_tmpa_int` is the number of successive occurrences of |

```

1738   \int_incr:N \l_tmpa_int
1739   \@@_patch_preamble_iii_i:n
1740 }

1741 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1742 {
1743   \str_if_eq:nnTF { #1 } |
1744   { \@@_patch_preamble_iii_i:n | }
1745   {
1746     \tl_gput_right:Nx \g_@@_preamble_tl
1747     {
1748       \exp_not:N !
1749       {
1750         \skip_horizontal:n
1751         {
1752           \dim_eval:n
1753           {
1754             \arrayrulewidth * \l_tmpa_int
1755             + \doublerulesep * ( \l_tmpa_int - 1)
1756           }
1757         }
1758       }
1759     }
1760     \tl_gput_right:Nx \g_@@_internal_code_after_tl
1761     { \@@_vline:nn { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } }
1762     \int_zero:N \l_tmpa_int
1763     \str_if_eq:nnT { #1 } { \q_stop }
1764     { \bool_gset_true:N \g_tmpb_bool }
1765     \@@_patch_preamble:n #1
1766   }
1767 }

1768 \bool_new:N \l_@@_bar_at_end_of_pream_bool

```

For p, m and b

```

1769 \cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
1770 {
1771   \tl_gput_right:Nn \g_@@_preamble_tl
1772   {
1773     > {
1774       \@@_Cell:
1775       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
1776       \mode_leave_vertical:
1777       \arraybackslash
1778       \vrule height \box_ht:N \@@_arstrutbox depth 0 pt width 0 pt
1779     }
1780     c
1781     < {
1782       \vrule height 0 pt depth \box_dp:N \@@_arstrutbox width 0 pt
1783       \end { minipage }
1784       \@@_end_Cell:
1785     }
1786   }

```

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

```

1787   \int_gincr:N \c@jCol
1788   \@@_patch_preamble_x:n
1789 }

```

For w and W

```

1790 \cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
1791 {
1792   \tl_gput_right:Nn \g_@@_preamble_tl

```



```

1793 {
1794   > {
1795     \hbox_set:Nw \l_@@_cell_box
1796     \@@_Cell:
1797     \tl_set:Nn \l_@@_cell_type_tl { #3 }
1798   }
1799   c
1800   < {
1801     \@@_end_Cell:
1802     #1
1803     \hbox_set_end:
1804     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1805     \@@_adjust_size_box:
1806     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
1807   }
1808 }

```

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

```

1809 \int_gincr:N \c@jCol
1810 \@@_patch_preamble_x:n
1811 }

```

For \@@_true_c: which will appear in our redefinition of the columns of type S (of siunitx).

```

1812 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
1813 {
1814   \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

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

```

1815 \int_gincr:N \c@jCol
1816 \@@_patch_preamble_x:n
1817 }

```

For (, [and \{.

```

1818 \cs_new_protected:Npn \@@_patch_preamble_vii:nn #1 #2
1819 {
1820   \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.

```

1821 \int_compare:nNnTF \c@jCol = \c_zero_int
1822 {
1823   \str_if_eq:VnTF \g_@@_left_delim_tl { . }
1824   {

```

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

```

1825     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1826     \tl_gset:Nn \g_@@_right_delim_tl { . }
1827     \@@_patch_preamble:n #2
1828   }
1829   {
1830     \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1831     \@@_patch_preamble_vii_i:nn { #1 } { #2 }
1832   }
1833 }
1834 { \@@_patch_preamble_vii_i:nn { #1 } { #2 } }
1835 }

1836 \cs_new_protected:Npn \@@_patch_preamble_vii_i:nn #1 #2
1837 {
1838   \tl_gput_right:Nx \g_@@_internal_code_after_tl
1839   { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
1840   \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
1841   {
1842     \@@_error:nn { delimiter~after~opening } { #2 }
1843     \@@_patch_preamble:n

```

```

1844     }
1845     { \@@_patch_preamble:n #2 }
1846 }

```

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}`).

```

1847 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
1848 {
1849     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
1850     \tl_if_in:nnTF { ) ] \} } { #2 }
1851     { \@@_patch_preamble_viii_i:nnn #1 #2 }
1852     {
1853         \tl_if_eq:nnTF { \q_stop } { #2 }
1854         {
1855             \str_if_eq:VnTF \g_@@_right_delim_tl { . }
1856             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
1857             {
1858                 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1859                 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1860                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1861                 \@@_patch_preamble:n #2
1862             }
1863         }
1864         {
1865             \tl_if_in:nnT { ( [ \{ } { #2 }
1866             { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
1867             \tl_gput_right:Nx \g_@@_internal_code_after_tl
1868             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1869             \@@_patch_preamble:n #2
1870         }
1871     }
1872 }
1873 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nnn #1 #2 #3
1874 {
1875     \tl_if_eq:nnTF { \q_stop } { #3 }
1876     {
1877         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
1878         {
1879             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1880             \tl_gput_right:Nx \g_@@_internal_code_after_tl
1881             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1882             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1883         }
1884         {
1885             \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1886             \tl_gput_right:Nx \g_@@_internal_code_after_tl
1887             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1888             \@@_error:nn { double-closing-delimiter } { #2 }
1889         }
1890     }
1891     {
1892         \tl_gput_right:Nx \g_@@_internal_code_after_tl
1893         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1894         \@@_error:nn { double-closing-delimiter } { #2 }
1895         \@@_patch_preamble:n #3
1896     }
1897 }
1898 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
1899 {
1900     \tl_gput_right:Nn \g_@@_preamble_tl

```

```
1901 { ! { \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`.

```
1902 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1903 { \@@_vdottedline:n { \int_use:N \c@jCol } }
1904 \@@_patch_preamble:n
1905 }
```

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.

```
1906 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
1907 {
1908   \str_if_eq:nnTF { #1 } { < }
1909   \@@_patch_preamble_ix:n
1910   {
1911     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1912     {
1913       \tl_gput_right:Nn \g_@@_preamble_tl
1914       { ! { \skip_horizontal:N \arrayrulewidth } }
1915     }
1916     {
1917       \exp_args:NNx
1918       \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
1919       {
1920         \tl_gput_right:Nn \g_@@_preamble_tl
1921         { ! { \skip_horizontal:N \arrayrulewidth } }
1922       }
1923     }
1924     \@@_patch_preamble:n { #1 }
1925   }
1926 }

1927 \cs_new_protected:Npn \@@_patch_preamble_ix:n #1
1928 {
1929   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
1930   \@@_patch_preamble_x:n
1931 }
```

17.1 The redefinition of `\multicolumn`

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

```
1932 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
1933 {
1934   \multispan { #1 }
1935   \begingroup
```

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

```
1936 \@temptokena = { #2 }
1937 \@tempswatrue
1938 \@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`.

```
1939 \tl_gclear:N \g_@@_preamble_tl
1940 \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
```

The following line creates the preamble. This line is a adaptation of a line of the initial definition of `\multicolumn`.

```
1941 \exp_args:NV \@mkpream \g_@@_preamble_tl
```

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

```

1942 \int_compare:nNt { #1 } > 1
1943 {
1944   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
1945   { \int_use:N \c@iRow - \@@_succ:n \c@jCol }
1946   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
1947   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
1948   {
1949     { \int_use:N \c@iRow }
1950     { \int_eval:n { \c@jCol + 1 } }
1951     { \int_use:N \c@iRow }
1952     { \int_eval:n { \c@jCol + #1 } }
1953   }
1954 }

```

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

```

1955 \cs_set:Npn \@sharp { #3 }
1956 \set@typeset@protect
1957 \cs_set_eq:NN \@startpbox \@@startpbox
1958 \cs_set_eq:NN \@endpbox \@@endpbox
1959 \@arstrut
1960 \@preamble
1961 \hbox { }
1962 \endgroup
1963 \int_gadd:Nn \c@jCol { #1 - 1 }
1964 \int_compare:nNt \c@jCol > \g_@@_col_total_int
1965 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1966 \ignorespaces
1967 }

```

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`.

```

1968 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
1969 {
1970   \str_case:nnF { #1 }
1971   {
1972     c { \@@_patch_m_preamble_i:n #1 }
1973     l { \@@_patch_m_preamble_i:n #1 }
1974     r { \@@_patch_m_preamble_i:n #1 }
1975     > { \@@_patch_m_preamble_ii:nn #1 }
1976     ! { \@@_patch_m_preamble_ii:nn #1 }
1977     @ { \@@_patch_m_preamble_ii:nn #1 }
1978     | { \@@_patch_m_preamble_iii:n #1 }
1979     p { \@@_patch_m_preamble_iv:nnn t #1 }
1980     m { \@@_patch_m_preamble_iv:nnn c #1 }
1981     b { \@@_patch_m_preamble_iv:nnn b #1 }
1982     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
1983     \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1984     \@@_true_c: { \@@_patch_m_preamble_vi:n #1 }
1985     C { \@@_error:nn { old~column~type } #1 }
1986     L { \@@_error:nn { old~column~type } #1 }
1987     R { \@@_error:nn { old~column~type } #1 }
1988     \q_stop { }
1989   }
1990   { \@@_fatal:nn { unknown~column~type } { #1 } }
1991 }

```

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

```

1992 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
1993 {
1994   \tl_gput_right:Nn \g_@@_preamble_tl

```

```

1995     {
1996     > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } }
1997     #1
1998     < \@@_end_Cell:
1999     }

```

We test for the presence of a <.

```

2000     \@@_patch_m_preamble_x:n
2001   }

```

For >, ! and @

```

2002 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2003 {
2004   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2005   \@@_patch_m_preamble:n
2006 }

```

For |

```

2007 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2008 {
2009   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2010   \@@_patch_m_preamble:n
2011 }

```

For p, m and b

```

2012 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2013 {
2014   \tl_gput_right:Nn \g_@@_preamble_tl
2015   {
2016     > {
2017       \@@_Cell:
2018       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2019       \mode_leave_vertical:
2020       \arraybackslash
2021       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2022     }
2023     c
2024     < {
2025       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2026       \end { minipage }
2027       \@@_end_Cell:
2028     }
2029   }

```

We test for the presence of a <.

```

2030     \@@_patch_m_preamble_x:n
2031   }

```

For w and W

```

2032 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2033 {
2034   \tl_gput_right:Nn \g_@@_preamble_tl
2035   {
2036     > {
2037       \hbox_set:Nw \l_@@_cell_box
2038       \@@_Cell:
2039       \tl_set:Nn \l_@@_cell_type_tl { #3 }
2040     }
2041     c
2042     < {
2043       \@@_end_Cell:
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 test for the presence of a <.

```

2051     \@@_patch_m_preamble_x:n
2052 }

```

For \@@_true_c: which will appear in our redefinition of the columns of type S (of siunitx).

```

2053 \cs_new_protected:Npn \@@_patch_m_preamble_vi:n #1
2054 {
2055     \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

We test for the presence of a <.

```

2056     \@@_patch_m_preamble_x:n
2057 }

```

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.

```

2058 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2059 {
2060     \str_if_eq:nnTF { #1 } { < }
2061     \@@_patch_m_preamble_ix:n
2062     {
2063         \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2064         {
2065             \tl_gput_right:Nn \g_@@_preamble_tl
2066             { ! { \skip_horizontal:N \arrayrulewidth } }
2067         }
2068         {
2069             \exp_args:NNx
2070             \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2071             {
2072                 \tl_gput_right:Nn \g_@@_preamble_tl
2073                 { ! { \skip_horizontal:N \arrayrulewidth } }
2074             }
2075         }
2076         \@@_patch_m_preamble:n { #1 }
2077     }
2078 }
2079 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2080 {
2081     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2082     \@@_patch_m_preamble_x:n
2083 }

```

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).

```

2084 \cs_new_protected:Npn \@@_put_box_in_flow:
2085 {
2086     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2087     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2088     \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2089     { \box_use_drop:N \l_tmpa_box }
2090     \@@_put_box_in_flow_i:
2091 }

```

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).

```

2092 \cs_new_protected:Npn \@@_put_box_in_flow_i:

```

```

2093 {
2094   \pgfpicture
2095     \@@_qpoint:n { row - 1 }
2096     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2097     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
2098     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2099     \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).

```

2100   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2101   {
2102     \int_set:Nn \l_tmpa_int
2103     {
2104       \str_range:Nnn
2105         \l_@@_baseline_tl
2106         6
2107         { \tl_count:V \l_@@_baseline_tl }
2108     }
2109     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2110   }
2111   {
2112     \str_case:VnF \l_@@_baseline_tl
2113     {
2114       { t } { \int_set:Nn \l_tmpa_int 1 }
2115       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2116     }
2117     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2118     \bool_lazy_or:nnT
2119     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2120     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2121     {
2122       \@@_error:n { bad~value~for~baseline }
2123       \int_set:Nn \l_tmpa_int 1
2124     }
2125     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2126     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2127   }
2128   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2129   \endpgfpicture
2130   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2131   \box_use_drop:N \l_tmpa_box
2132 }

```

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).

```

2133 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2134 {

```

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

```

2135   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2136   \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.

```

2137   \@@_create_extra_nodes:

```

```

2138 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2139 \bool_lazy_or:nnT
2140 { \int_compare_p:nNn \c@tabularnote > 0 }
2141 { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2142 \@@_insert_tabularnotes:
2143 \end { minipage }
2144 }
2145 \cs_new_protected:Npn \@@_insert_tabularnotes:
2146 {
2147 \skip_vertical:N 0.65ex

```

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

```

2148 \group_begin:
2149 \l_@@_notes_code_before_tl
2150 \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.

```

2151 \int_compare:nNnT \c@tabularnote > 0
2152 {
2153 \bool_if:NTF \l_@@_notes_para_bool
2154 {
2155 \begin { tabularnotes* }
2156 \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2157 \end { tabularnotes* }

```

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

```

2158 \par
2159 }
2160 {
2161 \tabularnotes
2162 \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2163 \endtabularnotes
2164 }
2165 }
2166 \unskip
2167 \group_end:
2168 \bool_if:NT \l_@@_notes_bottomrule_bool
2169 {
2170 \bool_if:NTF \c_@@_booktabs_loaded_bool
2171 {

```

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

```

2172 \skip_vertical:N \aboverulesep

```

`\CT@arc@` is the specification of color defined by `colortbl` but you use it even if `colortbl` is not loaded.

```

2173 { \CT@arc@ \hrule height \heavyrulewidth }
2174 }
2175 { \@@_error:n { bottomrule~without~booktabs } }
2176 }
2177 \l_@@_notes_code_after_tl
2178 \seq_gclear:N \g_@@_tabularnotes_seq
2179 \int_gzero:N \c@tabularnote
2180 }

```

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`.

```

2181 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2182 {
2183 \pgfpicture
2184 \@@_qpoint:n { row - 1 }
2185 \dim_gset_eq:NN \g_tmpa_dim \pgf@y

```



```

2186 \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2187 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2188 \endpgfpicture
2189 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2190 \int_compare:nNnT \l_@@_first_row_int = 0
2191 {
2192   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2193   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2194 }
2195 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2196 }

```

Now, the general case.

```

2197 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2198 {

```

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

```

2199 \tl_if_eq:NnT \l_@@_baseline_tl { t }
2200 { \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 .

```

2201 \pgfpicture
2202 \@@_qpoint:n { row - 1 }
2203 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2204 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2205 {
2206   \int_set:Nn \l_tmpa_int
2207   {
2208     \str_range:Nnn
2209     \l_@@_baseline_tl
2210     6
2211     { \tl_count:V \l_@@_baseline_tl }
2212   }
2213   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2214 }
2215 {
2216   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2217   \bool_lazy_or:nnT
2218   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2219   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2220   {
2221     \@@_error:n { bad-value-for-baseline }
2222     \int_set:Nn \l_tmpa_int 1
2223   }
2224   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2225 }
2226 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2227 \endpgfpicture
2228 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2229 \int_compare:nNnT \l_@@_first_row_int = 0
2230 {
2231   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2232   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2233 }
2234 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2235 }

```

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.

```

2236 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2237 {

```

We will compute the real width of both delimiters used.

```

2238 \dim_zero_new:N \l_@@_real_left_delim_dim
2239 \dim_zero_new:N \l_@@_real_right_delim_dim
2240 \hbox_set:Nn \l_tmpb_box
2241 {
2242   \c_math_toggle_token
2243   \left #1
2244   \vcenter
2245   {
2246     \vbox_to_ht:nn
2247     { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2248     { }
2249   }
2250   \right .
2251   \c_math_toggle_token
2252 }
2253 \dim_set:Nn \l_@@_real_left_delim_dim
2254 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2255 \hbox_set:Nn \l_tmpb_box
2256 {
2257   \c_math_toggle_token
2258   \left .
2259   \vbox_to_ht:nn
2260   { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2261   { }
2262   \right #2
2263   \c_math_toggle_token
2264 }
2265 \dim_set:Nn \l_@@_real_right_delim_dim
2266 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2267 \skip_horizontal:N \l_@@_left_delim_dim
2268 \skip_horizontal:N -\l_@@_real_left_delim_dim
2269 \@@_put_box_in_flow:
2270 \skip_horizontal:N \l_@@_right_delim_dim
2271 \skip_horizontal:N -\l_@@_real_right_delim_dim
2272 }

```

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).

```

2273 \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).

```

2274 {
2275   \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`).

```

2276   { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2277 }
2278 {
2279   \@@_create_col_nodes:
2280   \endarray
2281 }

```

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`).

```
2282 \NewDocumentEnvironment { @@-light-syntax } { b }
2283 {
```

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`.

```
2284 \tl_if_empty:nT { #1 } { @@_fatal:n { empty-environment } }
2285 \tl_map_inline:nn { #1 }
2286 {
2287   \str_if_eq:nnT { ##1 } { & }
2288   { @@_fatal:n { ampersand-in-light-syntax } }
2289   \str_if_eq:nnT { ##1 } { \ }
2290   { @@_fatal:n { double-backslash-in-light-syntax } }
2291 }
```

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`.

```
2292 @@_light_syntax_i #1 \CodeAfter \q_stop
2293 }
```

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`.

```
2294 { }
2295 \cs_new_protected:Npn @@_light_syntax_i #1\CodeAfter #2\q_stop
2296 {
2297   \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).

```
2298 \seq_gclear_new:N \g_@@_rows_seq
2299 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2300 \exp_args:NV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

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

```
2301 \int_compare:nNnT \l_@@_last_row_int = { -1 }
2302 { \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`).

```
2303 \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.

```
2304 \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2305 \exp_args:NV @@_line_with_light_syntax_i:n \l_tmpa_tl
2306 \seq_map_function:NN \g_@@_rows_seq @@_line_with_light_syntax:n
2307 @@_create_col_nodes:
2308 \endarray
2309 }
2310 \cs_new_protected:Npn @@_line_with_light_syntax:n #1
2311 { \tl_if_empty:nF { #1 } { \ \ @@_line_with_light_syntax_i:n { #1 } } }
2312 \cs_new_protected:Npn @@_line_with_light_syntax_i:n #1
2313 {
2314   \seq_gclear_new:N \g_@@_cells_seq
2315   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2316   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2317   \l_tmpa_tl
2318   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2319 }
```

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).

```

2320 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2321 {
2322   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2323   { \@@_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.

```

2324   \end { #2 }
2325 }

```

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).

```

2326 \cs_new:Npn \@@_create_col_nodes:
2327 {
2328   \crrcr
2329   \int_compare:nNnT \l_@@_first_col_int = 0
2330   {
2331     \omit
2332     \hbox_overlap_left:n
2333     {
2334       \bool_if:NT \l_@@_code_before_bool
2335       { \pgfsys@markposition { \@@_env: - col - 0 } }
2336       \pgfpicture
2337       \pgfrememberpicturerepositiononpagetrue
2338       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
2339       \str_if_empty:NF \l_@@_name_str
2340       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2341       \endpgfpicture
2342       \skip_horizontal:N 2\col@sep
2343       \skip_horizontal:N \g_@@_width_first_col_dim
2344     }
2345     &
2346   }
2347   \omit

```

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

```

2348   \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`).

```

2349   \int_compare:nNnTF \l_@@_first_col_int = 0
2350   {
2351     \bool_if:NT \l_@@_code_before_bool
2352     {
2353       \hbox
2354       {
2355         \skip_horizontal:N -0.5\arrayrulewidth
2356         \pgfsys@markposition { \@@_env: - col - 1 }
2357         \skip_horizontal:N 0.5\arrayrulewidth
2358       }
2359     }
2360     \pgfpicture
2361     \pgfrememberpicturerepositiononpagetrue
2362     \pgfcoordinate { \@@_env: - col - 1 }
2363     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2364     \str_if_empty:NF \l_@@_name_str
2365     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2366     \endpgfpicture
2367   }
2368   {

```

```

2369 \bool_if:NT \l_@@_code_before_bool
2370 {
2371     \hbox
2372     {
2373         \skip_horizontal:N 0.5\arrayrulewidth
2374         \pgfsys@markposition { \@@_env: - col - 1 }
2375         \skip_horizontal:N -0.5\arrayrulewidth
2376     }
2377 }
2378 \pgfpicture
2379 \pgfrememberpicturepositiononpagetrue
2380 \pgfcoordinate { \@@_env: - col - 1 }
2381 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
2382 \str_if_empty:NF \l_@@_name_str
2383 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2384 \endpgfpicture
2385 }

```

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.

```

2386 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2387 \bool_if:NF \l_@@_auto_columns_width_bool
2388 { \dim_compare:nNt \l_@@_columns_width_dim > \c_zero_dim }
2389 {
2390     \bool_lazy_and:nnTF
2391     \l_@@_auto_columns_width_bool
2392     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2393     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2394     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2395     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2396 }
2397 \skip_horizontal:N \g_tmpa_skip
2398 \hbox
2399 {
2400     \bool_if:NT \l_@@_code_before_bool
2401     {
2402         \hbox
2403         {
2404             \skip_horizontal:N -0.5\arrayrulewidth
2405             \pgfsys@markposition { \@@_env: - col - 2 }
2406             \skip_horizontal:N 0.5\arrayrulewidth
2407         }
2408     }
2409     \pgfpicture
2410     \pgfrememberpicturepositiononpagetrue
2411     \pgfcoordinate { \@@_env: - col - 2 }
2412     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2413     \str_if_empty:NF \l_@@_name_str
2414     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2415     \endpgfpicture
2416 }

```

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.

```

2417 \int_gset:Nn \g_tmpa_int 1
2418 \bool_if:NFT \g_@@_last_col_found_bool
2419 { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
2420 { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
2421 {
2422     &

```

```

2423 \omit
2424 \int_gincr:N \g_tmpa_int

```

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

```

2425 \skip_horizontal:N \g_tmpa_skip
2426 \bool_if:NT \l_@@_code_before_bool
2427 {
2428   \hbox
2429   {
2430     \skip_horizontal:N -0.5\arrayrulewidth
2431     \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2432     \skip_horizontal:N 0.5\arrayrulewidth
2433   }
2434 }

```

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

```

2435 \pgfpicture
2436 \pgfrememberpicturepositiononpagetrue
2437 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2438 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2439 \str_if_empty:NF \l_@@_name_str
2440 {
2441   \pgfnodealias
2442   { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2443   { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2444 }
2445 \endpgfpicture
2446 }

```

We want to remove the exterior `\arraycolsep` of a environment `{NiceArray}`.

```

2447 \bool_lazy_all:nT
2448 {
2449   \l_@@_NiceArray_bool
2450   { \bool_not_p:n \l_@@_NiceTabular_bool }
2451   { \clist_if_empty_p:N \l_@@_vlines_clist }
2452   { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2453   { ! \l_@@_bar_at_end_of_pream_bool }
2454 }
2455 { \skip_horizontal:n { - \col@sep } }

2456 \bool_if:NT \g_@@_last_col_found_bool
2457 {
2458   \hbox_overlap_right:n
2459   {
2460     \skip_horizontal:N \g_@@_width_last_col_dim
2461     \bool_if:NT \l_@@_code_before_bool
2462     {
2463       \pgfsys@markposition
2464       { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2465     }
2466     \pgfpicture
2467     \pgfrememberpicturepositiononpagetrue
2468     \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2469     \pgfpointorigin
2470     \str_if_empty:NF \l_@@_name_str
2471     {
2472       \pgfnodealias
2473       { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2474       { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2475     }
2476     \endpgfpicture
2477   }
2478 }

```

```

2479   \cr
2480 }

```

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

```

2481 \tl_const:Nn \c_@@_preamble_first_col_tl
2482 {
2483   >
2484   {

```

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

```

2485     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
2486     \bool_gset_true:N \g_@@_after_col_zero_bool
2487     \@@_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.

```

2488     \hbox_set:Nw \l_@@_cell_box
2489     \@@_math_toggle_token:
2490     \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”.

```

2491     \bool_lazy_and:nnT
2492     { \int_compare_p:nNn \c@iRow > 0 }
2493     {
2494       \bool_lazy_or_p:nn
2495       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2496       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2497     }
2498     {
2499       \l_@@_code_for_first_col_tl
2500       \xglobal \colorlet { nicematrix-first-col } { . }
2501     }
2502   }

```

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`.

```

2503   l
2504   <
2505   {
2506     \@@_math_toggle_token:
2507     \hbox_set_end:
2508     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2509     \@@_adjust_size_box:
2510     \@@_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.

```

2511     \dim_gset:Nn \g_@@_width_first_col_dim
2512     { \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.

```

2513     \hbox_overlap_left:n
2514     {
2515       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2516       \@@_node_for_cell:
2517       { \box_use_drop:N \l_@@_cell_box }
2518       \skip_horizontal:N \l_@@_left_delim_dim
2519       \skip_horizontal:N \l_@@_left_margin_dim
2520       \skip_horizontal:N \l_@@_extra_left_margin_dim
2521     }
2522     \bool_gset_false:N \g_@@_empty_cell_bool
2523     \skip_horizontal:N -2\col@sep

```

```

2524 }
2525 }

```

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

```

2526 \tl_const:Nn \c_@@_preamble_last_col_tl
2527 {
2528   >
2529   {

```

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

```

2530   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n

```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```

2531   \bool_gset_true:N \g_@@_last_col_found_bool
2532   \int_gincr:N \c@jCol
2533   \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.

```

2534   \hbox_set:Nw \l_@@_cell_box
2535   \@@_math_toggle_token:
2536   \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”.

```

2537   \int_compare:nNnT \c@iRow > 0
2538   {
2539     \bool_lazy_or:nnT
2540     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2541     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2542     {
2543       \l_@@_code_for_last_col_tl
2544       \xglobal \colorlet { nicematrix-last-col } { . }
2545     }
2546   }
2547 }
2548 \l
2549 <
2550 {
2551   \@@_math_toggle_token:
2552   \hbox_set_end:
2553   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2554   \@@_adjust_size_box:
2555   \@@_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.

```

2556   \dim_gset:Nn \g_@@_width_last_col_dim
2557   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2558   \skip_horizontal:N -2\col@sep

```

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

```

2559   \hbox_overlap_right:n
2560   {
2561     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2562     {
2563       \skip_horizontal:N \l_@@_right_delim_dim
2564       \skip_horizontal:N \l_@@_right_margin_dim
2565       \skip_horizontal:N \l_@@_extra_right_margin_dim
2566       \@@_node_for_cell:
2567     }
2568   }
2569   \bool_gset_false:N \g_@@_empty_cell_bool
2570 }
2571 }

```


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.

```

2572 \NewDocumentEnvironment { NiceArray } { }
2573 {
2574   \bool_set_true:N \l_@@_NiceArray_bool
2575   \str_if_empty:NT \g_@@_name_env_str
2576     { \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).

```

2577   \NiceArrayWithDelims . .
2578 }
2579 { \endNiceArrayWithDelims }

```

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

```

2580 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2581 {
2582   \NewDocumentEnvironment { #1 NiceArray } { }
2583   {
2584     \str_if_empty:NT \g_@@_name_env_str
2585       { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2586     \@@_test_if_math_mode:
2587     \NiceArrayWithDelims #2 #3
2588   }
2589   { \endNiceArrayWithDelims }
2590 }
2591 \@@_def_env:nnn p ( )
2592 \@@_def_env:nnn b [ ]
2593 \@@_def_env:nnn B \{ \}
2594 \@@_def_env:nnn v | |
2595 \@@_def_env:nnn V \| \|

```

The environment `{NiceMatrix}` and its variants

```

2596 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2597 {
2598   \bool_set_true:N \l_@@_Matrix_bool
2599   \use:c { #1 NiceArray }
2600   {
2601     *
2602     {
2603       \int_compare:nNnTF \l_@@_last_col_int < 0
2604         \c@MaxMatrixCols
2605         { \@@_pred:n \l_@@_last_col_int }
2606     }
2607     { > \@@_Cell: #2 < \@@_end_Cell: }
2608   }
2609 }
2610 \clist_map_inline:nn { { } , p , b , B , v , V }
2611 {
2612   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2613   {
2614     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2615     \tl_set:Nn \l_@@_type_of_col_tl c
2616     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2617     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2618   }
2619   { \use:c { end #1 NiceArray } }

```

```
2620 }
```

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

```
2621 \cs_new_protected:Npn \@@_NotEmpty:
2622 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

The environments `{NiceTabular}` and `{NiceTabular*}`

```
2623 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2624 {
2625   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2626   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2627   \bool_set_true:N \l_@@_NiceTabular_bool
2628   \NiceArray { #2 }
2629 }
2630 { \endNiceArray }

2631 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
2632 {
2633   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2634   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
2635   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2636   \bool_set_true:N \l_@@_NiceTabular_bool
2637   \NiceArray { #3 }
2638 }
2639 { \endNiceArray }
```

After the construction of the array

```
2640 \cs_new_protected:Npn \@@_after_array:
2641 {
2642   \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.

```
2643   \bool_if:NT \g_@@_last_col_found_bool
2644   { \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`.

```
2645   \bool_if:NT \l_@@_last_col_without_value_bool
2646   { \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.

```
2647   \bool_if:NT \l_@@_last_row_without_value_bool
2648   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

2649   \tl_gput_right:Nx \g_@@_aux_tl
2650   {
2651     \seq_gset_from_clist:Nn \exp_not:N \c_@@_size_seq
2652     {
2653       \int_use:N \l_@@_first_row_int ,
2654       \int_use:N \c_iRow ,
2655       \int_use:N \g_@@_row_total_int ,
2656       \int_use:N \l_@@_first_col_int ,
2657       \int_use:N \c_jCol ,
2658       \int_use:N \g_@@_col_total_int
2659     }

```

```

2660     \iow_newline:
2661 }

```

We write also the potential content of `\g_@@_pos_of_blocks_seq` (it will be useful if the command `\rowcolors` is used with the key `respect-blocks`).

```

2662 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
2663 {
2664   \tl_gput_right:Nx \g_@@_aux_tl
2665   {
2666     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
2667     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
2668     \iow_newline:
2669   }
2670 }
2671 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
2672 {
2673   \tl_gput_right:Nx \g_@@_aux_tl
2674   {
2675     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
2676     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
2677     \iow_newline:
2678     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
2679     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
2680     \iow_newline:
2681   }
2682 }

```

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

```

2683 \@@_create_diag_nodes:

```

By default, the diagonal lines will be parallelized⁵⁹. 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.

```

2684 \bool_if:NT \l_@@_parallelize_diags_bool
2685 {
2686   \int_gzero_new:N \g_@@_ddots_int
2687   \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```

2688   \dim_gzero_new:N \g_@@_delta_x_one_dim
2689   \dim_gzero_new:N \g_@@_delta_y_one_dim
2690   \dim_gzero_new:N \g_@@_delta_x_two_dim
2691   \dim_gzero_new:N \g_@@_delta_y_two_dim
2692 }
2693 \int_zero_new:N \l_@@_initial_i_int
2694 \int_zero_new:N \l_@@_initial_j_int
2695 \int_zero_new:N \l_@@_final_i_int
2696 \int_zero_new:N \l_@@_final_j_int
2697 \bool_set_false:N \l_@@_initial_open_bool
2698 \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_radius_dim` and `\l_@@_inter_dots_dim` (used to draw the dotted lines created by `\hdottedline` and `\vdotteline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.

```

2699 \bool_if:NT \l_@@_small_bool
2700 {
2701   \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
2702   \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }

```

⁵⁹It's possible to use the option `parallelize-diags` to disable this parallelization.

The dimension `\l_@@_xdots_shorten_dim` corresponds to the option `xdots/shorten` available to the user. That’s why we give a new value according to the current value, and not an absolute value.

```
2703     \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
2704 }
```

Now, we actually draw the dotted lines (specified by `\Cdots`, `\Vdots`, etc.).

```
2705 \@@_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.

```
2706 \@@_compute_corners:
```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```
2707 \@@_adjust_pos_of_blocks_seq:
```

The following code is only for efficiency. We determine whether the potential horizontal and vertical rules are “complete”, that is to say drawn in the whole array. We are sure that all the rules will be complete when there is no block, no virtual block (determined by a command such as `\Cdots`, `\Vdots`, etc.) and no corners. In that case, we switch to a shortcut version of `\@@_vline_i:nn` and `\@@_hline:nn`.

```
2708 \bool_lazy_all:nT
2709 {
2710   { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
2711   { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
2712   { \seq_if_empty_p:N \l_@@_corners_cells_seq }
2713 }
2714 {
2715   \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn
2716   \cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn
2717 }
2718 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
2719 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
2720 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
```

Now, the internal code-after and then, the `\CodeAfter`.

```
2721 \bool_if:NT \c_@@_tikz_loaded_bool
2722 {
2723   \tikzset
2724   {
2725     every-picture / .style =
2726     {
2727       overlay ,
2728       remember~picture ,
2729       name~prefix = \@@_env: -
2730     }
2731   }
2732 }
2733 \cs_set_eq:NN \line \@@_line
2734 \g_@@_internal_code_after_tl
2735 \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.

```
2736 \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).

```
2737 \seq_gclear:N \g_@@_submatrix_names_seq
```

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

```
2738 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```
2739 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
2740 \scan_stop:
2741 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
2742 \tl_gclear:N \g_nicematrix_code_after_tl
2743 \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.

```
2744 \tl_if_empty:NF \g_nicematrix_code_before_tl
2745 {
```

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`.

```
2746 \cs_set_protected:Npn \rectanglecolor { }
2747 \cs_set_protected:Npn \columncolor { }
2748 \tl_gput_right:Nx \g_@@_aux_tl
2749 {
2750   \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
2751     { \exp_not:N \g_nicematrix_code_before_tl }
2752   \iow_newline:
2753 }
2754 \bool_set_true:N \l_@@_code_before_bool
2755 }
2756 % \bool_if:NT \l_@@_code_before_bool \@@_write_aux_for_cell_nodes:

2757 \str_gclear:N \g_@@_name_env_str
2758 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁶⁰. 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`.

```
2759 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
2760 }
```

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`.

```
2761 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
2762 { \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).

```
2763 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
2764 {
```

⁶⁰e.g. `\color[rgb]{0.5,0.5,0}`

```

2765 \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
2766 { \@@_adjust_pos_of_blocks_seq_i:nnnn ##1 }
2767 }

```

The following command must *not* be protected.

```

2768 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnn #1 #2 #3 #4
2769 {
2770   { #1 }
2771   { #2 }
2772   {
2773     \int_compare:nNnTF { #3 } > { 99 }
2774     { \int_use:N \c@iRow }
2775     { #3 }
2776   }
2777   {
2778     \int_compare:nNnTF { #4 } > { 99 }
2779     { \int_use:N \c@jCol }
2780     { #4 }
2781   }
2782 }

```

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).

```

2783 \AtBeginDocument
2784 {
2785   \cs_new_protected:Npx \@@_draw_dotted_lines:
2786   {
2787     \c_@@_pgfortikzpicture_tl
2788     \@@_draw_dotted_lines_i:
2789     \c_@@_endpgfortikzpicture_tl
2790   }
2791 }

```

The following command *must* be protected because it will appear in the construction of the command `\@@_draw_dotted_lines:`.

```

2792 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
2793 {
2794   \pgfrememberpicturepositiononpagetrue
2795   \pgf@relevantforpicturesizefalse
2796   \g_@@_HVdotsfor_lines_tl
2797   \g_@@_Vdots_lines_tl
2798   \g_@@_Ddots_lines_tl
2799   \g_@@_Iddots_lines_tl
2800   \g_@@_Cdots_lines_tl
2801   \g_@@_Ldots_lines_tl
2802 }

2803 \cs_new_protected:Npn \@@_restore_iRow_jCol:
2804 {
2805   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
2806   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
2807 }

```

We define a new PGF shape for the diag nodes because we want to provide a anchor called `.5` for those nodes.

```

2808 \pgfdeclareshape { @@_diag_node }
2809 {
2810   \savedanchor { \five }
2811   {
2812     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
2813     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
2814   }

```

```

2815 \anchor { 5 } { \five }
2816 \anchor { center } { \pgfpictureorigin }
2817 }

2818 \cs_new_protected:Npn \@@_write_aux_for_cell_nodes:
2819 {
2820 \pgfpicture
2821 \pgfrememberpicturepositiononpagetrue
2822 \pgf@relevantforpicturesizefalse
2823 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2824 {
2825 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2826 {
2827 \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
2828 {
2829 \pgfscope
2830 \pgftransformshift
2831 { \pgfpictureanchor { \@@_env: - ##1 - #####1 } { north-west } }
2832 \pgfnode
2833 { rectangle }
2834 { center }
2835 {
2836 \hbox
2837 { \pgfsys@markposition { \@@_env: - ##1 - #####1 - NW } }
2838 }
2839 { }
2840 { }
2841 \endpgfscope
2842 \pgfscope
2843 \pgftransformshift
2844 { \pgfpictureanchor { \@@_env: - ##1 - #####1 } { south-east } }
2845 \pgfnode
2846 { rectangle }
2847 { center }
2848 {
2849 \hbox
2850 { \pgfsys@markposition { \@@_env: - ##1 - #####1 - SE } }
2851 }
2852 { }
2853 { }
2854 \endpgfscope
2855 }
2856 }
2857 }
2858 \endpgfpicture
2859 \@@_create_extra_nodes:
2860 }
2861 % \end{macrocode}
2862 %
2863 % \bigskip
2864 % The following command creates the diagonal nodes (in fact, if the matrix is
2865 % not a square matrix, not all the nodes are on the diagonal).
2866 % \begin{macrocode}
2867 \cs_new_protected:Npn \@@_create_diag_nodes:
2868 {
2869 \pgfpicture
2870 \pgfrememberpicturepositiononpagetrue
2871 \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
2872 {
2873 \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
2874 \dim_set_eq:NN \l_tmpa_dim \pgf@x
2875 \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
2876 \dim_set_eq:NN \l_tmpb_dim \pgf@y

```

```

2877 \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
2878 \dim_set_eq:NN \l_tmpc_dim \pgf@x
2879 \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
2880 \dim_set_eq:NN \l_tmpd_dim \pgf@y
2881 \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 `@â_diag_node`) that we will construct.

```

2882 \dim_set:Nn \l_tmpa_dim { ( \l_tmpc_dim - \l_tmpa_dim ) / 2 }
2883 \dim_set:Nn \l_tmpb_dim { ( \l_tmpd_dim - \l_tmpb_dim ) / 2 }
2884 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
2885 }

```

Now, the last node. Of course, that is only a coordinate because there is not `.5` anchor for that node.

```

2886 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
2887 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
2888 \dim_set_eq:NN \l_tmpa_dim \pgf@y
2889 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
2890 \pgfcoordinate
2891 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
2892 \pgfnodealias
2893 { \@@_env: - last }
2894 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
2895 \endpgfpicture
2896 }

```

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.

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

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```

2897 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
2898 {

```

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

```

2899 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```


Initialization of variables.

```

2900   \int_set:Nn \l_@@_initial_i_int { #1 }
2901   \int_set:Nn \l_@@_initial_j_int { #2 }
2902   \int_set:Nn \l_@@_final_i_int { #1 }
2903   \int_set:Nn \l_@@_final_j_int { #2 }

```

We will do two loops: one when determinating the initial cell and the other when determinating the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```

2904   \bool_set_false:N \l_@@_stop_loop_bool
2905   \bool_do_until:Nn \l_@@_stop_loop_bool
2906   {
2907     \int_add:Nn \l_@@_final_i_int { #3 }
2908     \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

2909     \bool_set_false:N \l_@@_final_open_bool
2910     \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
2911     {
2912       \int_compare:nNnTF { #3 } = 1
2913       { \bool_set_true:N \l_@@_final_open_bool }
2914       {
2915         \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
2916         { \bool_set_true:N \l_@@_final_open_bool }
2917       }
2918     }
2919     {
2920       \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
2921       {
2922         \int_compare:nNnT { #4 } = { -1 }
2923         { \bool_set_true:N \l_@@_final_open_bool }
2924       }
2925       {
2926         \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
2927         {
2928           \int_compare:nNnT { #4 } = 1
2929           { \bool_set_true:N \l_@@_final_open_bool }
2930         }
2931       }
2932     }
2933     \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.

```

2934     {

```

We do a step backwards.

```

2935       \int_sub:Nn \l_@@_final_i_int { #3 }
2936       \int_sub:Nn \l_@@_final_j_int { #4 }
2937       \bool_set_true:N \l_@@_stop_loop_bool
2938     }

```

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`.

```

2939     {
2940       \cs_if_exist:cTF
2941       {
2942         @@ _ dotted _
2943         \int_use:N \l_@@_final_i_int -
2944         \int_use:N \l_@@_final_j_int
2945       }
2946       {
2947         \int_sub:Nn \l_@@_final_i_int { #3 }
2948         \int_sub:Nn \l_@@_final_j_int { #4 }
2949         \bool_set_true:N \l_@@_final_open_bool
2950         \bool_set_true:N \l_@@_stop_loop_bool

```

```

2951     }
2952     {
2953         \cs_if_exist:cTF
2954         {
2955             pgf @ sh @ ns @ \l_@@_env:
2956             - \int_use:N \l_@@_final_i_int
2957             - \int_use:N \l_@@_final_j_int
2958         }
2959         { \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.

```

2960         {
2961             \cs_set:cpn
2962             {
2963                 @@ _ dotted _
2964                 \int_use:N \l_@@_final_i_int -
2965                 \int_use:N \l_@@_final_j_int
2966             }
2967             { }
2968         }
2969     }
2970 }
2971 }

```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programming is similar to the previous one.

```

2972     \bool_set_false:N \l_@@_stop_loop_bool
2973     \bool_do_until:Nn \l_@@_stop_loop_bool
2974     {
2975         \int_sub:Nn \l_@@_initial_i_int { #3 }
2976         \int_sub:Nn \l_@@_initial_j_int { #4 }
2977         \bool_set_false:N \l_@@_initial_open_bool
2978         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
2979         {
2980             \int_compare:nNnTF { #3 } = 1
2981             { \bool_set_true:N \l_@@_initial_open_bool }
2982             {
2983                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
2984                 { \bool_set_true:N \l_@@_initial_open_bool }
2985             }
2986         }
2987     {
2988         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
2989         {
2990             \int_compare:nNnT { #4 } = 1
2991             { \bool_set_true:N \l_@@_initial_open_bool }
2992         }
2993         {
2994             \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
2995             {
2996                 \int_compare:nNnT { #4 } = { -1 }
2997                 { \bool_set_true:N \l_@@_initial_open_bool }
2998             }
2999         }
3000     }
3001     \bool_if:NNTF \l_@@_initial_open_bool
3002     {
3003         \int_add:Nn \l_@@_initial_i_int { #3 }

```

```

3004         \int_add:Nn \l_@@_initial_j_int { #4 }
3005         \bool_set_true:N \l_@@_stop_loop_bool
3006     }
3007     {
3008         \cs_if_exist:cTF
3009         {
3010             @@ _ dotted _
3011             \int_use:N \l_@@_initial_i_int -
3012             \int_use:N \l_@@_initial_j_int
3013         }
3014         {
3015             \int_add:Nn \l_@@_initial_i_int { #3 }
3016             \int_add:Nn \l_@@_initial_j_int { #4 }
3017             \bool_set_true:N \l_@@_initial_open_bool
3018             \bool_set_true:N \l_@@_stop_loop_bool
3019         }
3020         {
3021             \cs_if_exist:cTF
3022             {
3023                 pgf @ sh @ ns @ \@@_env:
3024                 - \int_use:N \l_@@_initial_i_int
3025                 - \int_use:N \l_@@_initial_j_int
3026             }
3027             { \bool_set_true:N \l_@@_stop_loop_bool }
3028             {
3029                 \cs_set:cpn
3030                 {
3031                     @@ _ dotted _
3032                     \int_use:N \l_@@_initial_i_int -
3033                     \int_use:N \l_@@_initial_j_int
3034                 }
3035                 { }
3036             }
3037         }
3038     }
3039 }

```

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.

```

3040     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3041     {
3042         { \int_use:N \l_@@_initial_i_int }
3043         { \int_use:N \l_@@_initial_j_int }
3044         { \int_use:N \l_@@_final_i_int }
3045         { \int_use:N \l_@@_final_j_int }
3046     }
3047 }

```

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).

```

3048 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3049 {
3050     \int_set:Nn \l_@@_row_min_int 1
3051     \int_set:Nn \l_@@_col_min_int 1
3052     \int_set_eq:NN \l_@@_row_max_int \c{iRow}
3053     \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`.

```

3054     \seq_map_inline:Nn \g_@@_submatrix_seq
3055     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
3056 }

```

#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.

```

3057 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3058 {
3059   \bool_if:nT
3060   {
3061     \int_compare_p:n { #3 <= #1 }
3062     && \int_compare_p:n { #1 <= #5 }
3063     && \int_compare_p:n { #4 <= #2 }
3064     && \int_compare_p:n { #2 <= #6 }
3065   }
3066   {
3067     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3068     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3069     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3070     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3071   }
3072 }

3073 \cs_new_protected:Npn \@@_set_initial_coords:
3074 {
3075   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3076   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3077 }
3078 \cs_new_protected:Npn \@@_set_final_coords:
3079 {
3080   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3081   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3082 }
3083 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3084 {
3085   \pgfpointanchor
3086   {
3087     \@@_env:
3088     - \int_use:N \l_@@_initial_i_int
3089     - \int_use:N \l_@@_initial_j_int
3090   }
3091   { #1 }
3092   \@@_set_initial_coords:
3093 }
3094 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3095 {
3096   \pgfpointanchor
3097   {
3098     \@@_env:
3099     - \int_use:N \l_@@_final_i_int
3100     - \int_use:N \l_@@_final_j_int
3101   }
3102   { #1 }
3103   \@@_set_final_coords:
3104 }

3105 \cs_new_protected:Npn \@@_open_x_initial_dim:
3106 {
3107   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3108   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3109   {
3110     \cs_if_exist:cT
3111     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3112     {
3113       \pgfpointanchor
3114       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3115       { west }

```

```

3116         \dim_set:Nn \l_@@_x_initial_dim
3117         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3118     }
3119 }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

3120 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3121 {
3122     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3123     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3124     \dim_add:Nn \l_@@_x_initial_dim \col@sep
3125 }
3126 }
3127 \cs_new_protected:Npn \@@_open_x_final_dim:
3128 {
3129     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3130     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3131     {
3132         \cs_if_exist:cT
3133         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3134         {
3135             \pgfpointanchor
3136             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3137             { east }
3138             \dim_set:Nn \l_@@_x_final_dim
3139             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3140         }
3141     }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

3142 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3143 {
3144     \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3145     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3146     \dim_sub:Nn \l_@@_x_final_dim \col@sep
3147 }
3148 }

```

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.

```

3149 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3150 {
3151     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3152     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3153     {
3154         \@@_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.

```

3155 \group_begin:
3156     \int_compare:nNnTF { #1 } = 0
3157     { \color { nicematrix-first-row } }
3158     {

```

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.

```

3159         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3160         { \color { nicematrix-last-row } }
3161     }
3162     \keys_set:nn { NiceMatrix / xdots } { #3 }
3163     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3164     \@@_actually_draw_Ldots:
3165 \group_end:

```

```

3166     }
3167 }

```

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`.

```

3168 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3169 {
3170   \bool_if:NTF \l_@@_initial_open_bool
3171   {
3172     \@@_open_x_initial_dim:
3173     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3174     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3175   }
3176   { \@@_set_initial_coords_from_anchor:n { base-east } }
3177   \bool_if:NTF \l_@@_final_open_bool
3178   {
3179     \@@_open_x_final_dim:
3180     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3181     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3182   }
3183   { \@@_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 (?).

```

3184   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3185   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3186   \@@_draw_line:
3187 }

```

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.

```

3188 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3189 {
3190   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3191   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3192   {
3193     \@@_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.

```

3194   \group_begin:
3195   \int_compare:nNnTF { #1 } = 0
3196   { \color { nicematrix-first-row } }
3197   {

```

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.

```

3198       \int_compare:nNnT { #1 } = \l_@@_last_row_int
3199       { \color { nicematrix-last-row } }
3200   }
3201   \keys_set:nn { NiceMatrix / xdots } { #3 }

```

```

3202         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3203         \@@_actually_draw_Cdots:
3204     \group_end:
3205 }
3206 }

```

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.`

```

3207 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3208 {
3209     \bool_if:NTF \l_@@_initial_open_bool
3210     { \@@_open_x_initial_dim: }
3211     { \@@_set_initial_coords_from_anchor:n { mid-east } }
3212     \bool_if:NTF \l_@@_final_open_bool
3213     { \@@_open_x_final_dim: }
3214     { \@@_set_final_coords_from_anchor:n { mid-west } }
3215     \bool_lazy_and:nnTF
3216         \l_@@_initial_open_bool
3217         \l_@@_final_open_bool
3218     {
3219         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3220         \dim_set_eq:NN \l_tmpa_dim \pgf@y
3221         \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
3222         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3223         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3224     }
3225     {
3226         \bool_if:NT \l_@@_initial_open_bool
3227         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3228         \bool_if:NT \l_@@_final_open_bool
3229         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3230     }
3231     \@@_draw_line:
3232 }
3233 \cs_new_protected:Npn \@@_open_y_initial_dim:
3234 {
3235     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3236     \dim_set:Nn \l_@@_y_initial_dim
3237     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3238     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3239     {
3240         \cs_if_exist:cT
3241         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3242         {
3243             \pgfpointanchor
3244             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3245             { north }
3246             \dim_set:Nn \l_@@_y_initial_dim
3247             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3248         }
3249     }
3250 }

```

```

3251 \cs_new_protected:Npn \@@_open_y_final_dim:
3252 {
3253   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3254   \dim_set:Nn \l_@@_y_final_dim
3255   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3256   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3257   {
3258     \cs_if_exist:cT
3259     { \pgf@sh@ns@ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3260     {
3261       \pgfpointanchor
3262       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3263       { south }
3264       \dim_set:Nn \l_@@_y_final_dim
3265       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3266     }
3267   }
3268 }

```

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.

```

3269 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3270 {
3271   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3272   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3273   {
3274     \@@_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.

```

3275   \group_begin:
3276   \int_compare:nNnTF { #2 } = 0
3277   { \color { nicematrix-first-col } }
3278   {
3279     \int_compare:nNnT { #2 } = \l_@@_last_col_int
3280     { \color { nicematrix-last-col } }
3281   }
3282   \keys_set:nn { NiceMatrix / xdots } { #3 }
3283   \tl_if_empty:VF \l_@@_xdots_color_tl
3284   { \color { \l_@@_xdots_color_tl } }
3285   \@@_actually_draw_Vdots:
3286   \group_end:
3287 }
3288 }

```

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`.

```

3289 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3290 {

```

The boolean `\l_tmpa_bool` indicates whether the column is of type 1 or may be considered as if.

```

3291   \bool_set_false:N \l_tmpa_bool

```


First the case when the line is closed on both ends.

```

3292 \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3293 {
3294   \@@_set_initial_coords_from_anchor:n { south-west }
3295   \@@_set_final_coords_from_anchor:n { north-west }
3296   \bool_set:Nn \l_tmpa_bool
3297     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3298 }

```

Now, we try to determine whether the column is of type c or may be considered as if.

```

3299 \bool_if:NTF \l_@@_initial_open_bool
3300   \@@_open_y_initial_dim:
3301   { \@@_set_initial_coords_from_anchor:n { south } }
3302 \bool_if:NTF \l_@@_final_open_bool
3303   \@@_open_y_final_dim:
3304   { \@@_set_final_coords_from_anchor:n { north } }
3305 \bool_if:NTF \l_@@_initial_open_bool
3306 {
3307   \bool_if:NTF \l_@@_final_open_bool
3308   {
3309     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3310     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3311     \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3312     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3313     \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.

```

3314 \int_compare:nNnT \l_@@_last_col_int > { -2 }
3315 {
3316   \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3317   {
3318     \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3319     \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3320     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3321     \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3322   }
3323 }
3324 }
3325 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3326 }
3327 {
3328   \bool_if:NTF \l_@@_final_open_bool
3329   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3330 }

```

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.

```

3331 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3332 {
3333   \dim_set:Nn \l_@@_x_initial_dim
3334   {
3335     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3336     \l_@@_x_initial_dim \l_@@_x_final_dim
3337   }
3338   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3339 }
3340 }
3341 }
3342 \@@_draw_line:
3343 }

```

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.

```

3344 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3345 {
3346   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3347   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3348   {
3349     \@@_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.

```

3350     \group_begin:
3351     \keys_set:nn { NiceMatrix / xdots } { #3 }
3352     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3353     \@@_actually_draw_Ddots:
3354   \group_end:
3355 }
3356 }

```

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`.

```

3357 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3358 {
3359   \bool_if:NTF \l_@@_initial_open_bool
3360   {
3361     \@@_open_y_initial_dim:
3362     % \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3363     % \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3364     \@@_open_x_initial_dim:
3365   }
3366   { \@@_set_initial_coords_from_anchor:n { south-east } }
3367   \bool_if:NTF \l_@@_final_open_bool
3368   {
3369     % \@@_open_y_final_dim:
3370     % \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3371     \@@_open_x_final_dim:
3372     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3373   }
3374   { \@@_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.

```

3375   \bool_if:NT \l_@@_parallelize_diags_bool
3376   {
3377     \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).

```

3378     \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.

```

3379     {
3380         \dim_gset:Nn \g_@@_delta_x_one_dim
3381         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3382         \dim_gset:Nn \g_@@_delta_y_one_dim
3383         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3384     }

```

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`.

```

3385     {
3386         \dim_set:Nn \l_@@_y_final_dim
3387         {
3388             \l_@@_y_initial_dim +
3389             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3390             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3391         }
3392     }
3393 }
3394 \@@_draw_line:
3395 }

```

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.

```

3396 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3397 {
3398     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3399     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3400     {
3401         \@@_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.

```

3402     \group_begin:
3403     \keys_set:nn { NiceMatrix / xdots } { #3 }
3404     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3405     \@@_actually_draw_Iddots:
3406     \group_end:
3407 }
3408 }

```

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`.

```

3409 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3410 {
3411     \bool_if:NTF \l_@@_initial_open_bool
3412     {
3413         \@@_open_y_initial_dim:
3414         \@@_open_x_initial_dim:

```

```

3415     }
3416     { \@@_set_initial_coords_from_anchor:n { south-west } }
3417     \bool_if:NTF \l_@@_final_open_bool
3418     {
3419         \@@_open_y_final_dim:
3420         \@@_open_x_final_dim:
3421     }
3422     { \@@_set_final_coords_from_anchor:n { north-east } }
3423     \bool_if:NT \l_@@_parallelize_diags_bool
3424     {
3425         \int_gincr:N \g_@@_iddots_int
3426         \int_compare:nNnTF \g_@@_iddots_int = 1
3427         {
3428             \dim_gset:Nn \g_@@_delta_x_two_dim
3429             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3430             \dim_gset:Nn \g_@@_delta_y_two_dim
3431             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3432         }
3433         {
3434             \dim_set:Nn \l_@@_y_final_dim
3435             {
3436                 \l_@@_y_initial_dim +
3437                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3438                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3439             }
3440         }
3441     }
3442     \@@_draw_line:
3443 }

```

The actual instructions for drawing the dotted line 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`

```

3444 \cs_new_protected:Npn \@@_draw_line:
3445 {
3446     \pgfrememberpicturepositiononpagetrue
3447     \pgf@relevantforpicturesizefalse
3448     \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
3449     \@@_draw_standard_dotted_line:
3450     \@@_draw_non_standard_dotted_line:
3451 }

```

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.

```

3452 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
3453 {
3454     \begin { scope }
3455     \exp_args:No \@@_draw_non_standard_dotted_line:n
3456     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3457 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_non_standard_dotted_line:n` is, in fact, the list of options.

```

3458 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
3459 {
3460   \@@_draw_non_standard_dotted_line:nVV
3461   { #1 }
3462   \l_@@_xdots_up_tl
3463   \l_@@_xdots_down_tl
3464 }

3465 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:nnn #1 #2 #3
3466 {
3467   \draw
3468   [
3469     #1 ,
3470     shorten~> = \l_@@_xdots_shorten_dim ,
3471     shorten~< = \l_@@_xdots_shorten_dim ,
3472   ]
3473   ( \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).

```

3474   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3475   node [ sloped , below ] { $ \scriptstyle #3 $ }
3476   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3477 \end { scope }
3478 }
3479 \cs_generate_variant:Nn \@@_draw_non_standard_dotted_line:nnn { n V V }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real round dots).

```

3480 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3481 {
3482   \bool_lazy_and:nnF
3483   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3484   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3485   {
3486     \pgfscope
3487     \pgftransformshift
3488     {
3489       \pgfpointlineattime { 0.5 }
3490       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3491       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3492     }
3493     \pgftransformrotate
3494     {
3495       \fp_eval:n
3496       {
3497         atand
3498         (
3499           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3500           \l_@@_x_final_dim - \l_@@_x_initial_dim
3501         )
3502       }
3503     }
3504     \pgfnode
3505     { rectangle }
3506     { south }
3507     {
3508       \c_math_toggle_token
3509       \scriptstyle \l_@@_xdots_up_tl
3510       \c_math_toggle_token

```

```

3511     }
3512     { }
3513     { \pgfusepath { } }
3514     \pgfnode
3515     { rectangle }
3516     { north }
3517     {
3518         \c_math_toggle_token
3519         \scriptstyle \l_@@_xdots_down_tl
3520         \c_math_toggle_token
3521     }
3522     { }
3523     { \pgfusepath { } }
3524     \endpgfscope
3525 }
3526 \group_begin:

```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of `expl3` to compute this length.

```

3527     \dim_zero_new:N \l_@@_l_dim
3528     \dim_set:Nn \l_@@_l_dim
3529     {
3530         \fp_to_dim:n
3531         {
3532             sqrt
3533             (
3534                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3535                 +
3536                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3537             )
3538         }
3539     }

```

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.

```

3540     \bool_lazy_or:nnF
3541     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3542     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3543     \@@_draw_standard_dotted_line_i:
3544 \group_end:
3545 }
3546 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3547 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3548 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

3549     \bool_if:NTF \l_@@_initial_open_bool
3550     {
3551         \bool_if:NTF \l_@@_final_open_bool
3552         {
3553             \int_set:Nn \l_tmpa_int
3554             { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3555         }
3556         {
3557             \int_set:Nn \l_tmpa_int
3558             {
3559                 \dim_ratio:nn
3560                 { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3561                 \l_@@_inter_dots_dim
3562             }
3563         }

```

```

3564 }
3565 {
3566   \bool_if:NTF \l_@@_final_open_bool
3567   {
3568     \int_set:Nn \l_tmpa_int
3569     {
3570       \dim_ratio:nn
3571       { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3572       \l_@@_inter_dots_dim
3573     }
3574   }
3575   {
3576     \int_set:Nn \l_tmpa_int
3577     {
3578       \dim_ratio:nn
3579       { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3580       \l_@@_inter_dots_dim
3581     }
3582   }
3583 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

3584 \dim_set:Nn \l_tmpa_dim
3585 {
3586   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3587   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3588 }
3589 \dim_set:Nn \l_tmpb_dim
3590 {
3591   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3592   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3593 }

```

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`.

```

3594 \int_set:Nn \l_tmpb_int
3595 {
3596   \bool_if:NTF \l_@@_initial_open_bool
3597   { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3598   { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3599 }

```

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.

```

3600 \dim_gadd:Nn \l_@@_x_initial_dim
3601 {
3602   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3603   \dim_ratio:nn
3604   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3605   { 2 \l_@@_l_dim }
3606   * \l_tmpb_int
3607 }
3608 \dim_gadd:Nn \l_@@_y_initial_dim
3609 {
3610   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3611   \dim_ratio:nn
3612   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3613   { 2 \l_@@_l_dim }
3614   * \l_tmpb_int
3615 }
3616 \pgf@relevantforpicturesizefalse

```

```

3617 \int_step_inline:nnn 0 \l_tmpa_int
3618 {
3619   \pgfpathcircle
3620   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3621   { \l_@@_radius_dim }
3622   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3623   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3624 }
3625 \pgfusepathqfill
3626 }

```

User commands available in the new environments

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\AtBeginDocument` and the *arg spec* will be rescanned.

```

3627 \AtBeginDocument
3628 {
3629   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3630   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3631   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3632   {
3633     \int_compare:nNnTF \c@jCol = 0
3634     { \@@_error:nn { in~first~col } \Ldots }
3635     {
3636       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3637       { \@@_error:nn { in~last~col } \Ldots }
3638       {
3639         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
3640         { #1 , down = #2 , up = #3 }
3641       }
3642     }
3643     \bool_if:NF \l_@@_nullify_dots_bool
3644     { \phantom { \ensuremath { \@@_old_ldots } } }
3645     \bool_gset_true:N \g_@@_empty_cell_bool
3646   }

3647   \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3648   {
3649     \int_compare:nNnTF \c@jCol = 0
3650     { \@@_error:nn { in~first~col } \Cdots }
3651     {
3652       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3653       { \@@_error:nn { in~last~col } \Cdots }
3654       {
3655         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
3656         { #1 , down = #2 , up = #3 }
3657       }
3658     }
3659     \bool_if:NF \l_@@_nullify_dots_bool
3660     { \phantom { \ensuremath { \@@_old_cdots } } }
3661     \bool_gset_true:N \g_@@_empty_cell_bool
3662   }

```



```

3663 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
3664 {
3665   \int_compare:nNnTF \c@iRow = 0
3666   { \@@_error:nn { in~first~row } \Vdots }
3667   {
3668     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
3669     { \@@_error:nn { in~last~row } \Vdots }
3670     {
3671       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
3672       { #1 , down = #2 , up = #3 }
3673     }
3674   }
3675   \bool_if:NF \l_@@_nullify_dots_bool
3676   { \phantom { \ensuremath { \@@_old_vdots } } }
3677   \bool_gset_true:N \g_@@_empty_cell_bool
3678 }

3679 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
3680 {
3681   \int_case:nnF \c@iRow
3682   {
3683     0 { \@@_error:nn { in~first~row } \Ddots }
3684     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
3685   }
3686   {
3687     \int_case:nnF \c@jCol
3688     {
3689       0 { \@@_error:nn { in~first~col } \Ddots }
3690       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
3691     }
3692     {
3693       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3694       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
3695       { #1 , down = #2 , up = #3 }
3696     }
3697   }
3698 }
3699 \bool_if:NF \l_@@_nullify_dots_bool
3700 { \phantom { \ensuremath { \@@_old_ddots } } }
3701 \bool_gset_true:N \g_@@_empty_cell_bool
3702 }

3703 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
3704 {
3705   \int_case:nnF \c@iRow
3706   {
3707     0 { \@@_error:nn { in~first~row } \Iddots }
3708     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
3709   }
3710   {
3711     \int_case:nnF \c@jCol
3712     {
3713       0 { \@@_error:nn { in~first~col } \Iddots }
3714       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
3715     }
3716     {
3717       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3718       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
3719       { #1 , down = #2 , up = #3 }
3720     }
3721   }
3722   \bool_if:NF \l_@@_nullify_dots_bool

```

```

3723         { \phantom { \ensuremath { \@@_old_iddots } } }
3724         \bool_gset_true:N \g_@@_empty_cell_bool
3725     }
3726 }

```

End of the \AtBeginDocument.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

3727 \keys_define:nn { NiceMatrix / Ddots }
3728 {
3729     draw-first .bool_set:N = \l_@@_draw_first_bool ,
3730     draw-first .default:n = true ,
3731     draw-first .value_forbidden:n = true
3732 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

3733 \cs_new_protected:Npn \@@_Hspace:
3734 {
3735     \bool_gset_true:N \g_@@_empty_cell_bool
3736     \hspace
3737 }
3738 \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.

```

3739 \cs_new:Npn \@@_Hdotsfor:
3740 {
3741     \bool_lazy_and:nnTF
3742     { \int_compare_p:nNn \c@jCol = 0 }
3743     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
3744     {
3745         \bool_if:NTF \g_@@_after_col_zero_bool
3746         {
3747             \multicolumn { 1 } { c } { }
3748             \@@_Hdotsfor_i
3749         }
3750         { \@@_fatal:n { Hdotsfor~in~col~0 } }
3751     }
3752     {
3753         \multicolumn { 1 } { c } { }
3754         \@@_Hdotsfor_i
3755     }
3756 }

```

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:).

```

3757 \AtBeginDocument
3758 {
3759     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3760     \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.

```

3761     \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
3762     {
3763         \tl_gput_right:Nx \g_@@_Hdotsfor_lines_tl
3764         {
3765             \@@_Hdotsfor:nnnn
3766             { \int_use:N \c@iRow }
3767             { \int_use:N \c@jCol }

```

```

3768         { #2 }
3769         {
3770             #1 , #3 ,
3771             down = \exp_not:n { #4 } ,
3772             up = \exp_not:n { #5 }
3773         }
3774     }
3775     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
3776 }
3777 }

```

Enf of \AtBeginDocument.

```

3778 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
3779 {
3780     \bool_set_false:N \l_@@_initial_open_bool
3781     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

3782     \int_set:Nn \l_@@_initial_i_int { #1 }
3783     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

3784     \int_compare:nNnTF { #2 } = 1
3785     {
3786         \int_set:Nn \l_@@_initial_j_int 1
3787         \bool_set_true:N \l_@@_initial_open_bool
3788     }
3789     {
3790         \cs_if_exist:cTF
3791         {
3792             pgf @ sh @ ns @ \@@_env:
3793             - \int_use:N \l_@@_initial_i_int
3794             - \int_eval:n { #2 - 1 }
3795         }
3796         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
3797         {
3798             \int_set:Nn \l_@@_initial_j_int { #2 }
3799             \bool_set_true:N \l_@@_initial_open_bool
3800         }
3801     }
3802     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
3803     {
3804         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3805         \bool_set_true:N \l_@@_final_open_bool
3806     }
3807     {
3808         \cs_if_exist:cTF
3809         {
3810             pgf @ sh @ ns @ \@@_env:
3811             - \int_use:N \l_@@_final_i_int
3812             - \int_eval:n { #2 + #3 }
3813         }
3814         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
3815         {
3816             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3817             \bool_set_true:N \l_@@_final_open_bool
3818         }
3819     }
3820     \group_begin:
3821     \int_compare:nNnTF { #1 } = 0
3822     { \color { nicematrix-first-row } }
3823     {
3824         \int_compare:nNnT { #1 } = \g_@@_row_total_int

```

```

3825         { \color { nicematrix-last-row } }
3826     }
3827     \keys_set:nn { NiceMatrix / xdots } { #4 }
3828     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3829     \@@_actually_draw_Ldots:
3830     \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).

```

3831     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
3832     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
3833 }

3834 \AtBeginDocument
3835 {
3836     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3837     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3838     \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
3839     {
3840         \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
3841         {
3842             \@@_Vdotsfor:nnnn
3843             { \int_use:N \c@iRow }
3844             { \int_use:N \c@jCol }
3845             { #2 }
3846             {
3847                 #1 , #3 ,
3848                 down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
3849             }
3850         }
3851     }
3852 }

```

Enf of `\AtBeginDocument`.

```

3853 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
3854 {
3855     \bool_set_false:N \l_@@_initial_open_bool
3856     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

3857     \int_set:Nn \l_@@_initial_j_int { #2 }
3858     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it’s a bit more complicated.

```

3859     \int_compare:nNnTF #1 = 1
3860     {
3861         \int_set:Nn \l_@@_initial_i_int 1
3862         \bool_set_true:N \l_@@_initial_open_bool
3863     }
3864     {
3865         \cs_if_exist:cTF
3866         {
3867             pgf @ sh @ ns @ \@@_env:
3868             - \int_eval:n { #1 - 1 }
3869             - \int_use:N \l_@@_initial_j_int
3870         }
3871         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
3872         {
3873             \int_set:Nn \l_@@_initial_i_int { #1 }
3874             \bool_set_true:N \l_@@_initial_open_bool
3875         }
3876     }

```

```

3877 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
3878 {
3879   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3880   \bool_set_true:N \l_@@_final_open_bool
3881 }
3882 {
3883   \cs_if_exist:cTF
3884   {
3885     pgf @ sh @ ns @ \@@_env:
3886     - \int_eval:n { #1 + #3 }
3887     - \int_use:N \l_@@_final_j_int
3888   }
3889   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
3890   {
3891     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
3892     \bool_set_true:N \l_@@_final_open_bool
3893   }
3894 }
3895 \group_begin:
3896 \int_compare:nNnTF { #2 } = 0
3897 { \color { nicematrix-first-col } }
3898 {
3899   \int_compare:nNnT { #2 } = \g_@@_col_total_int
3900   { \color { nicematrix-last-col } }
3901 }
3902 \keys_set:nn { NiceMatrix / xdots } { #4 }
3903 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3904 \@@_actually_draw_Vdots:
3905 \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).

```

3906 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
3907 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
3908 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

3909 \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).⁶¹

```

3910 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
3911 { \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).

⁶¹Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

3912 \AtBeginDocument
3913 {
3914   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
3915   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3916   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
3917   {
3918     \group_begin:
3919     \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
3920     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3921     \use:e
3922     {
3923       \@@_line_i:nn
3924       { \@@_double_int_eval:n #2 \q_stop }
3925       { \@@_double_int_eval:n #3 \q_stop }
3926     }
3927     \group_end:
3928   }
3929 }
3930 \cs_new_protected:Npn \@@_line_i:nn #1 #2
3931 {
3932   \bool_set_false:N \l_@@_initial_open_bool
3933   \bool_set_false:N \l_@@_final_open_bool
3934   \bool_if:nTF
3935   {
3936     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
3937     ||
3938     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
3939   }
3940   {
3941     \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
3942   }
3943   { \@@_draw_line_ii:nn { #1 } { #2 } }
3944 }
3945 \AtBeginDocument
3946 {
3947   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
3948   {

```

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:`.

```

3949   \c_@@_pgfortikzpicture_tl
3950   \@@_draw_line_iii:nn { #1 } { #2 }
3951   \c_@@_endpgfortikzpicture_tl
3952 }
3953 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

3954 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
3955 {
3956   \pgfrememberpicturepositiononpagetrue
3957   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
3958   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3959   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3960   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
3961   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3962   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3963   \@@_draw_line:
3964 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That's why we try to draw rectangles of the same color in the same instruction `\pgfusepath { fill }` (and they will be in the same instruction `fill`—coded `f`— in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor` (which are linked to `\rowcolor`, `\columncolor` and `\rectanglecolor` before the execution of the `code-before`) 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 `\@@_rowcolor:n`, `\@@_columncolor:n` and `\@@_rectanglecolor:nn` (corresponding of `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor`).

`\bigskip #1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_color_seq` 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).

```
3965 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
3966 {
```

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.

```
3967   \int_zero:N \l_tmpa_int
3968   \seq_map_indexed_inline:Nn \g_@@_colors_seq
3969     { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
3970   \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

First, the case where the color is a *new* color (not in the sequence).

```
3971   {
3972     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
3973     \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
3974   }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
3975     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
3976   }
3977 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
```

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`).

```
3978 \cs_new_protected:Npn \@@_actually_color:
3979 {
3980   \pgfpicture
3981   \pgf@relevantforpicturesizefalse
3982   \seq_map_indexed_inline:Nn \g_@@_colors_seq
3983     {
3984       \color ##2
3985       \use:c { g_@@_color _ ##1 _ tl }
3986       \tl_gclear:c { g_@@_color _ ##1 _ tl }
3987       \pgfusepath { fill }
3988     }
3989   \endpgfpicture
3990 }
```

```

3991 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
3992 {
3993   \tl_set:Nn \l_tmpa_tl { #1 }
3994   \tl_set:Nn \l_tmpb_tl { #2 }
3995 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

3996 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
3997 {
3998   \tl_if_blank:nF { #2 }
3999   {
4000     \@@_add_to_colors_seq:xn
4001     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4002     { \@@_rowcolor:n { #3 } }
4003   }
4004 }
4005 \cs_new_protected:Npn \@@_rowcolor:n #1
4006 {
4007   \tl_set:Nn \l_@@_rows_tl { #1 }
4008   \tl_set:Nn \l_@@_cols_tl { - }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4009 \@@_cartesian_path:
4010 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

4011 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4012 {
4013   \tl_if_blank:nF { #2 }
4014   {
4015     \@@_add_to_colors_seq:xn
4016     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4017     { \@@_columncolor:n { #3 } }
4018   }
4019 }
4020 \cs_new_protected:Npn \@@_columncolor:n #1
4021 {
4022   \tl_set:Nn \l_@@_rows_tl { - }
4023   \tl_set:Nn \l_@@_cols_tl { #1 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4024 \@@_cartesian_path:
4025 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

4026 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4027 {
4028   \tl_if_blank:nF { #2 }
4029   {
4030     \@@_add_to_colors_seq:xn
4031     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4032     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4033   }
4034 }

```


The last argument is the radius of the corners of the rectangle.

```

4035 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4036 {
4037   \tl_if_blank:nF { #2 }
4038   {
4039     \@@_add_to_colors_seq:xn
4040     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4041     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4042   }
4043 }

```

The last argument is the radius of the corners of the rectangle.

```

4044 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4045 {
4046   \@@_cut_on_hyphen:w #1 \q_stop
4047   \tl_clear_new:N \l_tmpc_tl
4048   \tl_clear_new:N \l_tmpd_tl
4049   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
4050   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
4051   \@@_cut_on_hyphen:w #2 \q_stop
4052   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
4053   \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`.

```

4054   \@@_cartesian_path:n { #3 }
4055 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

4056 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4057 {
4058   \clist_map_inline:nn { #3 }
4059   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4060 }

```

```

4061 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4062 {
4063   \int_step_inline:nn { \int_use:N \c@iRow }
4064   {
4065     \int_step_inline:nn { \int_use:N \c@jCol }
4066     {
4067       \int_if_even:nTF { ####1 + ##1 }
4068       { \@@_cellcolor [ #1 ] { #2 } }
4069       { \@@_cellcolor [ #1 ] { #3 } }
4070     } { ##1 - ####1 }
4071   }
4072 }
4073 }

```

```

4074 \keys_define:nn { NiceMatrix / arraycolor }
4075 { except-corners .code:n = \@@_error:n { key except-corners } }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns). The third argument is a optional argument which a list of pairs key-value.

```

4076 \NewDocumentCommand \@@_arraycolor { 0 { } m 0 { } }
4077 {
4078   \keys_set:nn { NiceMatrix / arraycolor } { #3 }
4079   \@@_rectanglecolor [ #1 ] { #2 }
4080   { 1 - 1 }
4081   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4082 }

```

```

4083 \keys_define:nn { NiceMatrix / rowcolors }
4084 {
4085   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4086   respect-blocks .default:n = true ,
4087   cols .tl_set:N = \l_@@_cols_tl ,
4088   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4089   restart .default:n = true ,
4090   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4091 }

```

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 first color ; #4 is the second color ; #5 is for the optional list of pairs key-value.

```

4092 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4093 {

```

The group is for the options.

```

4094   \group_begin:
4095   \tl_clear_new:N \l_@@_cols_tl
4096   \tl_set:Nn \l_@@_cols_tl { - }
4097   \keys_set:nn { NiceMatrix / rowcolors } { #5 }

```

The boolean `\l_tmpa_bool` will indicate whereas we are in a row of the first color or of the second color.

```

4098   \bool_set_true:N \l_tmpa_bool
4099   \bool_if:NT \l_@@_respect_blocks_bool
4100   {

```

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`).

```

4101     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4102     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
4103     { \@@_not_in_exterior_p:nnnn ##1 }
4104   }
4105   \pgfpicture
4106   \pgf@relevantforpicturesizefalse
4107   \clist_map_inline:nn { #2 }
4108   {
4109     \tl_set:Nn \l_tmpa_tl { ##1 }
4110     \tl_if_in:NnTF \l_tmpa_tl { - }
4111     { \@@_cut_on_hyphen:w ##1 \q_stop }
4112     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c{iRow } } }

```

The counter `\l_tmpa_int` will be the index of the loop.

```

4113     \int_set:Nn \l_tmpa_int \l_tmpa_tl
4114     \bool_if:NNTF \l_@@_rowcolors_restart_bool
4115     { \bool_set_true:N \l_tmpa_bool }
4116     { \bool_set:Nn \l_tmpa_bool { \int_if_odd_p:n { \l_tmpa_tl } } }
4117     \int_zero_new:N \l_tmpc_int
4118     \int_set:Nn \l_tmpc_int \l_tmpb_tl
4119     \int_do_until:nNnn \l_tmpa_int > \l_tmpc_int
4120     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4121     \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).

```

4122     \bool_if:NT \l_@@_respect_blocks_bool
4123     {
4124       \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
4125       { \@@_intersect_our_row_p:nnnn #####1 }
4126       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4127     }
4128     \tl_set:Nx \l_@@_rows_tl
4129     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
4130     \bool_if:NTF \l_tmpa_bool
4131     {
4132         \tl_if_blank:nF { #3 }
4133         {
4134             \tl_if_empty:nTF { #1 }
4135             \color
4136             { \color [ #1 ] }
4137             { #3 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4138         \@@_cartesian_path:
4139         \pgfusepath { fill }
4140     }
4141     \bool_set_false:N \l_tmpa_bool
4142 }
4143 {
4144     \tl_if_blank:nF { #4 }
4145     {
4146         \tl_if_empty:nTF { #1 }
4147         \color
4148         { \color [ #1 ] }
4149         { #4 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

4150         \@@_cartesian_path:
4151         \pgfusepath { fill }
4152     }
4153     \bool_set_true:N \l_tmpa_bool
4154 }
4155     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4156 }
4157 }
4158 \endpgfpicture
4159 \group_end:
4160 }

```

```

4161 \cs_new_protected:Npn \@@_rowcolors_i:nnnn #1 #2 #3 #4
4162 {
4163     \int_compare:nNnT { #3 } > \l_tmpb_int
4164     { \int_set:Nn \l_tmpb_int { #3 } }
4165 }

```

```

4166 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnn p
4167 {
4168     \bool_lazy_or:nnTF
4169     { \int_compare_p:nNn { #4 } = \c_zero_int }
4170     { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
4171     \prg_return_false:
4172     \prg_return_true:
4173 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

4174 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnn p
4175 {
4176     \bool_if:nTF

```

```

4177 {
4178   \int_compare_p:n { #1 <= \l_tmpa_int }
4179   &&
4180   \int_compare_p:n { \l_tmpa_int <= #3 }
4181 }
4182 \prg_return_true:
4183 \prg_return_false:
4184 }

```

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`).

```

4185 \cs_new_protected:Npn \@@_cartesian_path:n #1
4186 {
4187   \bool_lazy_and:nnT
4188   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4189   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4190   {
4191     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4192     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4193   }

```

We begin the loop over the columns.

```

4194   \clist_map_inline:Nn \l_@@_cols_tl
4195   {
4196     \tl_set:Nn \l_tmpa_tl { ##1 }
4197     \tl_if_in:NnTF \l_tmpa_tl { - }
4198     { \@@_cut_on_hyphen:w ##1 \q_stop }
4199     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4200     \bool_lazy_or:nnT
4201     { \tl_if_blank_p:V \l_tmpa_tl }
4202     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4203     { \tl_set:Nn \l_tmpa_tl { 1 } }
4204     \bool_lazy_or:nnT
4205     { \tl_if_blank_p:V \l_tmpb_tl }
4206     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4207     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4208     \int_compare:nNnT \l_tmpb_tl > \c@jCol
4209     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_tmpc_tl` will contain the number of column.

```

4210     \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.

```

4211     \@@_qpoint:n { col - \l_tmpa_tl }
4212     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4213     { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4214     { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4215     \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4216     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4217   \clist_map_inline:Nn \l_@@_rows_tl
4218   {
4219     \tl_set:Nn \l_tmpa_tl { #####1 }
4220     \tl_if_in:NnTF \l_tmpa_tl { - }
4221     { \@@_cut_on_hyphen:w #####1 \q_stop }
4222     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4223     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }

```

```

4224 \tl_if_empty:NT \l_tmpb_tl
4225 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4226 \int_compare:nNnT \l_tmpb_tl > \c@iRow
4227 { \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.
4228 \seq_if_in:NxF \l_@@_corners_cells_seq
4229 { \l_tmpa_tl - \l_tmpc_tl }
4230 {
4231 \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
4232 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4233 \@@_qpoint:n { row - \l_tmpa_tl }
4234 \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4235 \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4236 \pgfpathrectanglecorners
4237 { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4238 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4239 }
4240 }
4241 }
4242 }

```

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`).

```

4243 \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.

```

4244 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4245 {
4246 \clist_set_eq:NN \l_tmpa_clist #1
4247 \clist_clear:N #1
4248 \clist_map_inline:Nn \l_tmpa_clist
4249 {
4250 \tl_set:Nn \l_tmpa_tl { ##1 }
4251 \tl_if_in:NnTF \l_tmpa_tl { - }
4252 { \@@_cut_on_hyphen:w ##1 \q_stop }
4253 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4254 \bool_lazy_or:nnT
4255 { \tl_if_blank_p:V \l_tmpa_tl }
4256 { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4257 { \tl_set:Nn \l_tmpa_tl { 1 } }
4258 \bool_lazy_or:nnT
4259 { \tl_if_blank_p:V \l_tmpb_tl }
4260 { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4261 { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4262 \int_compare:nNnT \l_tmpb_tl > #2
4263 { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4264 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4265 { \clist_put_right:Nn #1 { ####1 } }
4266 }
4267 }

```

When the user uses the key `colortbl`-like, the following command will be linked to `\cellcolor` in the tabular.

```

4268 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4269 {
4270 \peek_remove_spaces:n
4271 {
4272 \tl_gput_right:Nx \g_nicematrix_code_before_tl
4273 {

```

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).

```

4274         \cellcolor [ #1 ] { \exp_not:n { #2 } }
4275         { \int_use:N \c@iRow - \int_use:N \c@jCol }
4276     }
4277 }
4278 }

```

When the user uses the key colortbl-like, the following command will be linked to \rowcolor in the tabular.

```

4279 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4280 {
4281     \peek_remove_spaces:n
4282     {
4283         \tl_gput_right:Nx \g_nicematrix_code_before_tl
4284         {
4285             \exp_not:N \rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4286             { \int_use:N \c@iRow - \int_use:N \c@jCol }
4287             { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4288         }
4289     }
4290 }

```

```

4291 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4292 {

```

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).

```

4293     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4294     {

```

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).

```

4295         \tl_gput_left:Nx \g_nicematrix_code_before_tl
4296         {
4297             \exp_not:N \columncolor [ #1 ]
4298             { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4299         }
4300     }
4301 }

```

The vertical rules

We give to the user the possibility to define new types of columns (with \newcolumnntype of array) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command \OnlyMainNiceMatrix in that goal. However, that command must be no-op outside the environments of nicematrix (and so the user will be allowed to use the same new type of column in the environments of nicematrix and in the standard environments of array). That's why we provide first a global definition of \OnlyMainNiceMatrix.

```

4302 \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.

```

4303 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4304 {
4305     \int_compare:nNnTF \l_@@_first_col_int = 0

```

```

4306 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4307 {
4308   \int_compare:nNnTF \c@jCol = 0
4309   {
4310     \int_compare:nNnF \c@iRow = { -1 }
4311     { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4312   }
4313   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4314 }
4315 }

```

This definition may seem complicated by we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell.

The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```

4316 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4317 {
4318   \int_compare:nNnF \c@iRow = 0
4319   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4320 }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to `-2` or `-1` (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the column `#1` (that is to say on the left side). `#2` is the number of consecutive occurrences of `|`.

```

4321 \cs_new_protected:Npn \@@_vline:nn #1 #2
4322 {

```

The following test is for the case where the user don't use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

4323   \int_compare:nNnT { #1 } < { \c@jCol + 2 }
4324   {
4325     \pgfpicture
4326     \@@_vline_i:nn { #1 } { #2 }
4327     \endpgfpicture
4328   }
4329 }
4330 \cs_new_protected:Npn \@@_vline_i:nn #1 #2
4331 {

```

`\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_tmppc_tl`.

```

4332   \tl_set:Nx \l_tmpb_tl { #1 }
4333   \tl_clear_new:N \l_tmppc_tl
4334   \int_step_variable:nNn \c@iRow \l_tmpa_tl
4335   {

```

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.

```

4336     \bool_gset_true:N \g_tmpa_bool
4337     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4338     { \@@_test_vline_in_block:nnnn ##1 }
4339     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4340     { \@@_test_vline_in_block:nnnn ##1 }
4341     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4342     { \@@_test_vline_in_stroken_block:nnnn ##1 }
4343     \clist_if_empty:NF \l_@@_corners_clist
4344     \@@_test_in_corner_v:
4345     \bool_if:NTF \g_tmpa_bool
4346     {
4347       \tl_if_empty:NT \l_tmppc_tl

```

We keep in memory that we have a rule to draw.

```

4348         { \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl }
4349     }
4350     {
4351         \tl_if_empty:NF \l_tmpc_tl
4352         {
4353             \@@_vline_ii:nnnn
4354             { #1 }
4355             { #2 }
4356             \l_tmpc_tl
4357             { \int_eval:n { \l_tmpa_tl - 1 } }
4358             \tl_clear:N \l_tmpc_tl
4359         }
4360     }
4361 }
4362 \tl_if_empty:NF \l_tmpc_tl
4363 {
4364     \@@_vline_ii:nnnn
4365     { #1 }
4366     { #2 }
4367     \l_tmpc_tl
4368     { \int_use:N \c@iRow }
4369     \tl_clear:N \l_tmpc_tl
4370 }
4371 }

4372 \cs_new_protected:Npn \@@_test_in_corner_v:
4373 {
4374     \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
4375     {
4376         \seq_if_in:NxT
4377         \l_@@_corners_cells_seq
4378         { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4379         { \bool_set_false:N \g_tmpa_bool }
4380     }
4381     {
4382         \seq_if_in:NxT
4383         \l_@@_corners_cells_seq
4384         { \l_tmpa_tl - \l_tmpb_tl }
4385         {
4386             \int_compare:nNnTF \l_tmpb_tl = 1
4387             { \bool_set_false:N \g_tmpa_bool }
4388             {
4389                 \seq_if_in:NxT
4390                 \l_@@_corners_cells_seq
4391                 { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4392                 { \bool_set_false:N \g_tmpa_bool }
4393             }
4394         }
4395     }
4396 }

```

#1 is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the numbers of the rows between which the rule has to be drawn.

```

4397 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
4398 {
4399     \pgfrememberpicturepositiononpagetrue
4400     \pgf@relevantforpicturesizefalse
4401     \@@_qpoint:n { row - #3 }
4402     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4403     \@@_qpoint:n { col - #1 }
4404     \dim_set_eq:NN \l_tmpb_dim \pgf@x

```



```

4405 \@@_qpoint:n { row - \@@_succ:n { #4 } }
4406 \dim_set_eq:NN \l_tmpc_dim \pgf@y
4407 \bool_lazy_and:nnT
4408 { \int_compare_p:nNn { #2 } > 1 }
4409 { ! \tl_if_blank_p:V \CT@drsc@ }
4410 {
4411   \group_begin:
4412   \CT@drsc@
4413   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4414   \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
4415   \dim_set:Nn \l_tmpd_dim
4416     { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4417   \pgfpathrectanglecorners
4418     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4419     { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4420   \pgfusepath { fill }
4421   \group_end:
4422 }
4423 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4424 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4425 \prg_replicate:nn { #2 - 1 }
4426 {
4427   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4428   \dim_sub:Nn \l_tmpb_dim \doublerulesep
4429   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4430   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4431 }
4432 \CT@arc@
4433 \pgfsetlinewidth { 1.1 \arrayrulewidth }
4434 \pgfsetrectcap
4435 \pgfusepathqstroke
4436 }

```

The following command draws a complete vertical rule in the column #1 (#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key `corners` is not used).

```

4437 \cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
4438 { \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

4439 \cs_new_protected:Npn \@@_draw_vlines:
4440 {
4441   \int_step_inline:nnn
4442   {
4443     \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4444       1 2
4445   }
4446   {
4447     \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4448       { \@@_succ:n \c@jCol }
4449       \c@jCol
4450   }
4451   {
4452     \tl_if_eq:NnF \l_@@_vlines_clist { all }
4453     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4454     { \@@_vline:nn { ##1 } 1 }
4455   }
4456 }

```

The horizontal rules

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the row #1. #2 is the number of consecutive occurrences of `\Hline`.

```

4457 \cs_new_protected:Npn \@@_hline:nn #1 #2
4458 {
4459   \pgfpicture
4460   \@@_hline_i:nn { #1 } { #2 }
4461   \endpgfpicture
4462 }
4463 \cs_new_protected:Npn \@@_hline_i:nn #1 #2
4464 {

```

`\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`.

```

4465   \tl_set:Nn \l_tmpa_tl { #1 }
4466   \tl_clear_new:N \l_tmpc_tl
4467   \int_step_variable:nNn \c@jCol \l_tmpb_tl
4468   {

```

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.

```

4469     \bool_gset_true:N \g_tmpa_bool
4470     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4471     { \@@_test_hline_in_block:nnnn ##1 }
4472     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4473     { \@@_test_hline_in_block:nnnn ##1 }
4474     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4475     { \@@_test_hline_in_stroken_block:nnnn ##1 }
4476     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4477     \bool_if:NTF \g_tmpa_bool
4478     {
4479       \tl_if_empty:NT \l_tmpc_tl

```

We keep in memory that we have a rule to draw.

```

4480       { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
4481     }
4482     {
4483       \tl_if_empty:NF \l_tmpc_tl
4484       {
4485         \@@_hline_ii:nnnn
4486         { #1 }
4487         { #2 }
4488         \l_tmpc_tl
4489         { \int_eval:n { \l_tmpb_tl - 1 } }
4490         \tl_clear:N \l_tmpc_tl
4491       }
4492     }
4493   }
4494   \tl_if_empty:NF \l_tmpc_tl
4495   {
4496     \@@_hline_ii:nnnn
4497     { #1 }
4498     { #2 }
4499     \l_tmpc_tl
4500     { \int_use:N \c@jCol }
4501     \tl_clear:N \l_tmpc_tl
4502   }
4503 }

```

```

4504 \cs_new_protected:Npn \@@_test_in_corner_h:
4505 {
4506   \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
4507   {
4508     \seq_if_in:NxT
4509     \l_@@_corners_cells_seq
4510     { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4511     { \bool_set_false:N \g_tmpa_bool }
4512   }
4513   {
4514     \seq_if_in:NxT
4515     \l_@@_corners_cells_seq
4516     { \l_tmpa_tl - \l_tmpb_tl }
4517     {
4518       \int_compare:nNnTF \l_tmpa_tl = 1
4519       { \bool_set_false:N \g_tmpa_bool }
4520       {
4521         \seq_if_in:NxT
4522         \l_@@_corners_cells_seq
4523         { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4524         { \bool_set_false:N \g_tmpa_bool }
4525       }
4526     }
4527   }
4528 }

```

#1 is the number of the row; #2 is the number of horizontal rules to draw (with potentially a color between); #3 and #4 are the number of the columns between which the rule has to be drawn.

```

4529 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
4530 {
4531   \pgfrememberpicturepositiononpagetrue
4532   \pgf@relevantforpicturesizefalse
4533   \@@_qpoint:n { col - #3 }
4534   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4535   \@@_qpoint:n { row - #1 }
4536   \dim_set_eq:NN \l_tmpb_dim \pgf@y
4537   \@@_qpoint:n { col - \@@_succ:n { #4 } }
4538   \dim_set_eq:NN \l_tmpc_dim \pgf@x
4539   \bool_lazy_and:nnT
4540   { \int_compare_p:nNn { #2 } > 1 }
4541   { ! \tl_if_blank_p:V \CT@drsc@ }
4542   {
4543     \group_begin:
4544     \CT@drsc@
4545     \dim_set:Nn \l_tmpd_dim
4546     { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4547     \pgfpathrectanglecorners
4548     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4549     { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4550     \pgfusepathqfill
4551     \group_end:
4552   }
4553   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4554   \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4555   \prg_replicate:nn { #2 - 1 }
4556   {
4557     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4558     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4559     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4560     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
4561   }
4562   \CT@arc@
4563   \pgfsetlinewidth { 1.1 \arrayrulewidth }

```

```

4564 \pgfsetrectcap
4565 \pgfusepathqstroke
4566 }

```

```

4567 \cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
4568 { \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as `\Cdots` and in the corners (if the key `corners` is used).

```

4569 \cs_new_protected:Npn \@@_draw_hlines:
4570 {
4571   \int_step_inline:nnn
4572   {
4573     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4574     1 2
4575   }
4576   {
4577     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4578     { \@@_succ:n \c@iRow }
4579     \c@iRow
4580   }
4581   {
4582     \tl_if_eq:NnF \l_@@_hlines_clist { all }
4583     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
4584     { \@@_hline:nn { ##1 } 1 }
4585   }
4586 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

4587 \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.

```

4588 \cs_set:Npn \@@_Hline_i:n #1
4589 {
4590   \peek_meaning_ignore_spaces:NTF \Hline
4591   { \@@_Hline_ii:nn { #1 + 1 } }
4592   { \@@_Hline_iii:n { #1 } }
4593 }
4594 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
4595 \cs_set:Npn \@@_Hline_iii:n #1
4596 {
4597   \skip_vertical:n
4598   {
4599     \arrayrulewidth * ( #1 )
4600     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
4601   }
4602   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4603   { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
4604   \ifnum 0 = ` { \fi }
4605 }

```

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`.

```

4606 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4

```

```

4607 {
4608   \bool_lazy_all:nT
4609   {
4610     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
4611     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4612     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4613     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4614   }
4615   { \bool_gset_false:N \g_tmpa_bool }
4616 }

```

The same for vertical rules.

```

4617 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4
4618 {
4619   \bool_lazy_all:nT
4620   {
4621     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4622     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4623     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
4624     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4625   }
4626   { \bool_gset_false:N \g_tmpa_bool }
4627 }
4628 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
4629 {
4630   \bool_lazy_all:nT
4631   {
4632     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4633     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
4634     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4635     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4636   }
4637   { \bool_gset_false:N \g_tmpa_bool }
4638 }
4639 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
4640 {
4641   \bool_lazy_all:nT
4642   {
4643     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4644     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4645     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4646     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
4647   }
4648   { \bool_gset_false:N \g_tmpa_bool }
4649 }

```

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.

```

4650 \cs_new_protected:Npn \@@_compute_corners:
4651 {

```

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.

```

4652   \seq_clear_new:N \l_@@_corners_cells_seq
4653   \clist_map_inline:Nn \l_@@_corners_clist
4654   {
4655     \str_case:nnF { ##1 }
4656     {
4657       { NW }

```

```

4658         { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
4659         { NE }
4660         { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
4661         { SW }
4662         { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
4663         { SE }
4664         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
4665     }
4666     { \@@_error:nn { bad~corner } { ##1 } }
4667 }

```

Even if the user has used the key `corners` (or the key `hvlines-except-corners`), the list of cells in the corners may be empty.

```

4668     \seq_if_empty:NF \l_@@_corners_cells_seq
4669     {

```

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.

```

4670         \tl_gput_right:Nx \g_@@_aux_tl
4671         {
4672             \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
4673             { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
4674             \iow_newline:
4675         }
4676     }
4677 }

```

“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.

```

4678 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
4679 {

```

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.

```

4680     \bool_set_false:N \l_tmpa_bool
4681     \int_zero_new:N \l_@@_last_empty_row_int
4682     \int_set:Nn \l_@@_last_empty_row_int { #1 }
4683     \int_step_inline:nnnn { #1 } { #3 } { #5 }
4684     {
4685         \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
4686         \bool_lazy_or:nnTF
4687         {
4688             \cs_if_exist_p:c
4689             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
4690         }
4691         \l_tmpb_bool
4692         { \bool_set_true:N \l_tmpa_bool }
4693         {
4694             \bool_if:NF \l_tmpa_bool
4695             { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
4696         }
4697     }

```

Now, you determine the last empty cell in the row of number 1.

```

4698 \bool_set_false:N \l_tmpa_bool
4699 \int_zero_new:N \l_@@_last_empty_column_int
4700 \int_set:Nn \l_@@_last_empty_column_int { #2 }
4701 \int_step_inline:nnnn { #2 } { #4 } { #6 }
4702 {
4703   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
4704   \bool_lazy_or:nnTF
4705     \l_tmpb_bool
4706     {
4707       \cs_if_exist_p:c
4708         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
4709     }
4710     { \bool_set_true:N \l_tmpa_bool }
4711     {
4712       \bool_if:NF \l_tmpa_bool
4713         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
4714     }
4715 }

```

Now, we loop over the rows.

```

4716 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
4717 {

```

We treat the row number ##1 with another loop.

```

4718   \bool_set_false:N \l_tmpa_bool
4719   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
4720   {
4721     \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
4722     \bool_lazy_or:nnTF
4723       \l_tmpb_bool
4724       {
4725         \cs_if_exist_p:c
4726           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
4727       }
4728       { \bool_set_true:N \l_tmpa_bool }
4729       {
4730         \bool_if:NF \l_tmpa_bool
4731         {
4732           \int_set:Nn \l_@@_last_empty_column_int { #####1 }
4733           \seq_put_right:Nn
4734             \l_@@_corners_cells_seq
4735             { ##1 - #####1 }
4736         }
4737       }
4738     }
4739   }
4740 }

```

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`).

```

4741 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
4742 {
4743   \int_set:Nn \l_tmpa_int { #1 }
4744   \int_set:Nn \l_tmpb_int { #2 }
4745   \bool_set_false:N \l_tmpb_bool
4746   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4747     { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
4748 }
4749 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6
4750 {
4751   \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }

```

```

4752 {
4753   \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
4754   {
4755     \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
4756     {
4757       \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
4758       { \bool_set_true:N \l_tmpb_bool }
4759     }
4760   }
4761 }
4762 }

```

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`.

```

4763 \cs_new:Npn \@@_hdottedline:
4764 {
4765   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
4766   \@@_hdottedline_i:
4767 }

```

On the other side, the following command should be protected.

```

4768 \cs_new_protected:Npn \@@_hdottedline_i:
4769 {

```

We write in the code-after the instruction that will actually draw the dotted line. It's not possible to draw this dotted line now because we don't know the length of the line (we don't even know the number of columns).

```

4770   \tl_gput_right:Nx \g_@@_internal_code_after_tl
4771   { \@@_hdottedline:n { \int_use:N \c@iRow } }
4772 }

```

The command `\@@_hdottedline:n` is the command written in the `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```

4773 \AtBeginDocument
4774 {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That's why we construct now a version of `\@@_hdottedline:n` with the right environment (`\begin{pgfpicture}\end{pgfpicture}` or `\begin{tikzpicture}...\end{tikzpicture}`).

```

4775   \cs_new_protected:Npx \@@_hdottedline:n #1
4776   {
4777     \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
4778     \bool_set_true:N \exp_not:N \l_@@_final_open_bool
4779     \c_@@_pgfortikzpicture_tl
4780     \@@_hdottedline_i:n { #1 }
4781     \c_@@_endpgfortikzpicture_tl
4782   }
4783 }

```

The following command *must* be protected since it is used in the construction of `\@@_hdottedline:n`.

```

4784 \cs_new_protected:Npn \@@_hdottedline_i:n #1
4785 {
4786   \pgfrememberpicturepositiononpagetrue
4787   \@@_qpoint:n { row - #1 }

```


We do a translation par `-l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```

4788 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4789 \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
4790 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim

```

The dotted line will be extended if the user uses `margin` (or `left-margin` and `right-margin`).

The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn’t).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

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

```

4791 \@@_qpoint:n { col - 1 }
4792 \dim_set:Nn \l_@@_x_initial_dim
4793 {
4794 \pgf@x +

```

We do a reduction by `\arraycolsep` for the environments with delimiters (and not for the other).

```

4795 \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4796 - \l_@@_left_margin_dim
4797 }
4798 \@@_qpoint:n { col - \@@_succ:n \c@jCol }
4799 \dim_set:Nn \l_@@_x_final_dim
4800 {
4801 \pgf@x -
4802 \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4803 + \l_@@_right_margin_dim
4804 }

```

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.

```

4805 \tl_if_eq:NnF \g_@@_left_delim_tl (
4806 { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
4807 \tl_if_eq:NnF \g_@@_right_delim_tl )
4808 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }

```

Up to now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

4809 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4810 \@@_draw_line:
4811 }

```

Vertical dotted lines

```

4812 \cs_new_protected:Npn \@@_vdottedline:n #1
4813 {
4814 \bool_set_true:N \l_@@_initial_open_bool
4815 \bool_set_true:N \l_@@_final_open_bool

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”.

```

4816 \bool_if:NTF \c_@@_tikz_loaded_bool
4817 {
4818   \tikzpicture
4819   \@@_vdottedline_i:n { #1 }
4820   \endtikzpicture
4821 }
4822 {
4823   \pgfpicture
4824   \@@_vdottedline_i:n { #1 }
4825   \endpgfpicture
4826 }
4827 }

4828 \cs_new_protected:Npn \@@_vdottedline_i:n #1
4829 {

```

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.

```

4830 \CT@arc@
4831 \pgfrememberpicturepositiononpagetrue
4832 \@@_qpoint:n { col - \int_eval:n { #1 + 1 } }

```

We do a translation par `-\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```

4833 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
4834 \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
4835 \@@_qpoint:n { row - 1 }

```

We arbitrary decrease the height of the dotted line by a quantity equal to `\l_@@_inter_dots_dim` in order to improve the visual impact.

```

4836 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
4837 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
4838 \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }

```

Up to now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```

4839 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4840 \@@_draw_line:
4841 }

```

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.

```

4842 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

4843 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
4844 {
4845   auto-columns-width .code:n =
4846   {
4847     \bool_set_true:N \l_@@_block_auto_columns_width_bool
4848     \dim_gzero_new:N \g_@@_max_cell_width_dim
4849     \bool_set_true:N \l_@@_auto_columns_width_bool
4850   }
4851 }

```

```

4852 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
4853 {
4854   \int_gincr:N \g_@@_NiceMatrixBlock_int
4855   \dim_zero:N \l_@@_columns_width_dim
4856   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
4857   \bool_if:NT \l_@@_block_auto_columns_width_bool
4858   {
4859     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4860     {
4861       \exp_args:Nnc \dim_set:Nn \l_@@_columns_width_dim
4862       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
4863     }
4864   }
4865 }

```

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).

```

4866 {
4867   \bool_if:NT \l_@@_block_auto_columns_width_bool
4868   {
4869     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
4870     \iow_shipout:Nx \@mainaux
4871     {
4872       \cs_gset:cpn
4873       { @@_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.

```

4874       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
4875     }
4876     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
4877   }
4878 }

```

The extra nodes

First, two variants of the functions \dim_min:nn and \dim_max:nn.

```

4879 \cs_generate_variant:Nn \dim_min:nn { v n }
4880 \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).

```

4881 \cs_new_protected:Npn \@@_create_extra_nodes:
4882 {
4883   \bool_if:nTF \l_@@_medium_nodes_bool
4884   {
4885     \bool_if:NTF \l_@@_large_nodes_bool
4886     \@@_create_medium_and_large_nodes:
4887     \@@_create_medium_nodes:
4888   }
4889   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
4890 }

```

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`.

```

4891 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
4892 {
4893   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4894   {
4895     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
4896     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
4897     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
4898     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
4899   }
4900   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4901   {
4902     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
4903     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
4904     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
4905     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
4906   }

```

We begin the two nested loops over the rows and the columns of the array.

```

4907   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4908   {
4909     \int_step_variable:nnNn
4910     \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.

```

4911     {
4912       \cs_if_exist:cT
4913       { 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`.

```

4914     {
4915       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
4916       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
4917       { \dim_min:vn { l_@@_row\_@@_i: _min_dim } \pgf@y }
4918       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
4919       {
4920         \dim_set:cn { l_@@_column\_@@_j: _min_dim }
4921         { \dim_min:vn { l_@@_column\_@@_j: _min_dim } \pgf@x }
4922       }

```

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`.

```

4923       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
4924       \dim_set:cn { l_@@_row\_@@_i: _max_dim }
4925       { \dim_max:vn { l_@@_row\_@@_i: _max_dim } \pgf@y }
4926       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
4927       {
4928         \dim_set:cn { l_@@_column\_@@_j: _max_dim }
4929         { \dim_max:vn { l_@@_column\_@@_j: _max_dim } \pgf@x }
4930       }
4931     }
4932   }
4933 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

4934 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
4935 {
4936   \dim_compare:nNnT
4937     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
4938     {
4939       \@@_qpoint:n { row - \@@_i: - base }
4940       \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
4941       \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
4942     }
4943 }
4944 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
4945 {
4946   \dim_compare:nNnT
4947     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
4948     {
4949       \@@_qpoint:n { col - \@@_j: }
4950       \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
4951       \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
4952     }
4953 }
4954 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

4955 \cs_new_protected:Npn \@@_create_medium_nodes:
4956 {
4957   \pgfpicture
4958     \pgfrememberpicturepositiononpagetrue
4959     \pgf@relevantforpicturesizefalse
4960     \@@_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”.

```

4961     \tl_set:Nn \l_@@_suffix_tl { -medium }
4962     \@@_create_nodes:
4963   \endpgfpicture
4964 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁶². 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:`.

```

4965 \cs_new_protected:Npn \@@_create_large_nodes:
4966 {
4967   \pgfpicture
4968     \pgfrememberpicturepositiononpagetrue
4969     \pgf@relevantforpicturesizefalse
4970     \@@_computations_for_medium_nodes:
4971     \@@_computations_for_large_nodes:
4972     \tl_set:Nn \l_@@_suffix_tl { -large }
4973     \@@_create_nodes:
4974   \endpgfpicture
4975 }
4976 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
4977 {

```

⁶²If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

4978 \pgfpicture
4979 \pgfrememberpicturepositiononpagetrue
4980 \pgf@relevantforpicturesizefalse
4981 \@@_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”.

```

4982 \tl_set:Nn \l_@@_suffix_tl { - medium }
4983 \@@_create_nodes:
4984 \@@_computations_for_large_nodes:
4985 \tl_set:Nn \l_@@_suffix_tl { - large }
4986 \@@_create_nodes:
4987 \endpgfpicture
4988 }

```

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.

```

4989 \cs_new_protected:Npn \@@_computations_for_large_nodes:
4990 {
4991 \int_set:Nn \l_@@_first_row_int 1
4992 \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`.

```

4993 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
4994 {
4995 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
4996 {
4997 (
4998 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
4999 \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5000 )
5001 / 2
5002 }
5003 \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5004 { l_@@_row _ \@@_i: _ min _ dim }
5005 }
5006 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5007 {
5008 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5009 {
5010 (
5011 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5012 \dim_use:c
5013 { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5014 )
5015 / 2
5016 }
5017 \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5018 { l_@@_column _ \@@_j: _ max _ dim }
5019 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

5020 \dim_sub:cn
5021 { l_@@_column _ 1 _ min _ dim }
5022 \l_@@_left_margin_dim
5023 \dim_add:cn
5024 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5025 \l_@@_right_margin_dim
5026 }

```

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 $\backslash l_@@_suffix_tl$ (-medium or -large).

```

5027 \cs_new_protected:Npn \@@_create_nodes:
5028 {
5029   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5030   {
5031     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5032     {

```

We draw the rectangular node for the cell ($\backslash@@_i$ - $\backslash@@_j$).

```

5033       \@@_pgf_rect_node:nnnnn
5034       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5035       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
5036       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
5037       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
5038       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
5039       \str_if_empty:NF \l_@@_name_str
5040       {
5041         \pgfnodealias
5042         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5043         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5044       }
5045     }
5046   }

```

Now, we create the nodes for the cells of the $\backslash multicolumn$. We recall that we have stored in $\backslash g_@@_multicolumn_cells_seq$ the list of the cells where a $\backslash multicolumn\{n\}\{\dots\}\{\dots\}$ with $n > 1$ was issued and in $\backslash g_@@_multicolumn_sizes_seq$ the correspondent values of n .

```

5047   \seq_mapthread_function:NNN
5048   \g_@@_multicolumn_cells_seq
5049   \g_@@_multicolumn_sizes_seq
5050   \@@_node_for_multicolumn:nn
5051 }

```

```

5052 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5053 {
5054   \cs_set_nopar:Npn \@@_i: { #1 }
5055   \cs_set_nopar:Npn \@@_j: { #2 }
5056 }

```

The command $\backslash@@_node_for_multicolumn:nn$ takes two arguments. The first is the position of the cell where the command $\backslash multicolumn\{n\}\{\dots\}\{\dots\}$ was issued in the format i - j and the second is the value of n (the length of the “multi-cell”).

```

5057 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5058 {
5059   \@@_extract_coords_values: #1 \q_stop
5060   \@@_pgf_rect_node:nnnnn
5061   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5062   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
5063   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
5064   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
5065   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
5066   \str_if_empty:NF \l_@@_name_str
5067   {
5068     \pgfnodealias
5069     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5070     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
5071   }
5072 }

```

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.

```

5073 \keys_define:nn { NiceMatrix / Block / FirstPass }
5074 {
5075   l .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
5076   l .value_forbidden:n = true ,
5077   r .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
5078   r .value_forbidden:n = true ,
5079   c .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
5080   c .value_forbidden:n = true ,
5081   L .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
5082   L .value_forbidden:n = true ,
5083   R .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
5084   R .value_forbidden:n = true ,
5085   C .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
5086   C .value_forbidden:n = true ,
5087   t .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl t ,
5088   t .value_forbidden:n = true ,
5089   b .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl b ,
5090   b .value_forbidden:n = true ,
5091   color .tl_set:N = \l_@@_color_tl ,
5092   color .value_required:n = true ,
5093 }

```

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 `>` (for TeX instructions put before the math mode of the label and before the beginning of the small array of the block). It's mandatory to use an expandable command.

```

5094 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
5095 {

```

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).

```

5096   \peek_remove_spaces:n
5097   {
5098     \tl_if_blank:nTF { #2 }
5099     { \@@_Block_i 1-1 \q_stop }
5100     { \@@_Block_i #2 \q_stop }
5101     { #1 } { #3 } { #4 }
5102   }
5103 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5104 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

Now, the arguments have been extracted: `#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of key-values, `#4` are the tokens to put before the math mode and the beginning of the small array of the block and `#5` is the label of the block.

```

5105 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5106 {

```

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).


```

5107 \bool_lazy_or:nnTF
5108   { \tl_if_blank_p:n { #1 } }
5109   { \str_if_eq_p:nn { #1 } { * } }
5110   { \int_set:Nn \l_tmpa_int { 100 } }
5111   { \int_set:Nn \l_tmpa_int { #1 } }
5112 \bool_lazy_or:nnTF
5113   { \tl_if_blank_p:n { #2 } }
5114   { \str_if_eq_p:nn { #2 } { * } }
5115   { \int_set:Nn \l_tmpb_int { 100 } }
5116   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5117 \int_compare:nNnTF \l_tmpb_int = 1
5118 {
5119   \tl_if_empty:NTF \l_@@_cell_type_tl
5120     { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5121     { \tl_set_eq:NN \l_@@_hpos_of_block_tl \l_@@_cell_type_tl }
5122 }
5123 { \tl_set:Nn \l_@@_hpos_of_block_tl c }

```

The value of `\l_@@_hpos_of_block_tl` may be modified by the keys of the command `\Block` that we will analyze now.

```

5124 \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
5125 \tl_set:Nx \l_tmpa_tl
5126 {
5127   { \int_use:N \c@iRow }
5128   { \int_use:N \c@jCol }
5129   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5130   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5131 }

```

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).

```

5132 \bool_lazy_or:nnTF
5133   { \int_compare_p:nNn { \l_tmpa_int } = 1 }
5134   { \int_compare_p:nNn { \l_tmpb_int } = 1 }
5135   { \exp_args:Nxx \@@_Block_iv:nnnnn }
5136   { \exp_args:Nxx \@@_Block_v:nnnnn }
5137 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5138 }

```

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).

```

5139 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5140 {
5141   \int_gincr:N \g_@@_block_box_int
5142   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5143   {
5144     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5145     {
5146       \@@_actually_diagbox:nnnnnn
5147       { \int_use:N \c@iRow }
5148       { \int_use:N \c@jCol }
5149       { \int_eval:n { \c@iRow + #1 - 1 } }

```

```

5150         { \int_eval:n { \c@jCol + #2 - 1 } }
5151         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5152     }
5153 }
5154 \box_gclear_new:c
5155 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5156 \hbox_gset:cn
5157 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5158 {

```

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`: because that command seems to be bugged: it doesn't work in XeLaTeX when `fontspec` is loaded.

```

5159     \tl_if_empty:NTF \l_@@_color_tl
5160     { \int_compare:nNnT { #2 } = 1 \set@color }
5161     { \color { \l_@@_color_tl } }
5162 \group_begin:
5163 \cs_set:Npn \arraystretch { 1 }
5164 \dim_set_eq:NN \extrarowheight \c_zero_dim
5165 #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 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).

```

5166     \bool_if:NT \g_@@_rotate_bool { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5167     \bool_if:NTF \l_@@_NiceTabular_bool
5168     {
5169         \use:x
5170         {
5171             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5172             { @ { } \l_@@_hpos_of_block_tl @ { } }
5173         }
5174         #5
5175     \end { tabular }
5176     }
5177     {
5178         \c_math_toggle_token
5179         \use:x
5180         {
5181             \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5182             { @ { } \l_@@_hpos_of_block_tl @ { } }
5183         }
5184         #5
5185     \end { array }
5186     \c_math_toggle_token
5187     }
5188 \group_end:
5189 }
5190 \bool_if:NT \g_@@_rotate_bool
5191 {
5192     \box_grotate:cn
5193     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5194     { 90 }
5195     \bool_gset_false:N \g_@@_rotate_bool
5196 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5197     \int_compare:nNnT { #2 } = 1
5198     {
5199         \dim_gset:Nn \g_@@_blocks_wd_dim

```

```

5200     {
5201         \dim_max:nn
5202         \g_@@_blocks_wd_dim
5203         {
5204             \box_wd:c
5205             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5206         }
5207     }
5208 }

```

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.

```

5209     \int_compare:nNnT { #1 } = 1
5210     {
5211         \dim_gset:Nn \g_@@_blocks_ht_dim
5212         {
5213             \dim_max:nn
5214             \g_@@_blocks_ht_dim
5215             {
5216                 \box_ht:c
5217                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5218             }
5219         }
5220         \dim_gset:Nn \g_@@_blocks_dp_dim
5221         {
5222             \dim_max:nn
5223             \g_@@_blocks_dp_dim
5224             {
5225                 \box_dp:c
5226                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5227             }
5228         }
5229     }
5230     \seq_gput_right:Nx \g_@@_blocks_seq
5231     {
5232         \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_of_block_tl. 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_of_block_tl, which is fixed by the type of current column.

```

5233         { \exp_not:n { #3 } , \l_@@_hpos_of_block_tl }
5234     {
5235         \box_use_drop:c
5236         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5237     }
5238 }
5239 }

```

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.

```

5240 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5241 {
5242     \seq_gput_right:Nx \g_@@_blocks_seq
5243     {
5244         \l_tmpa_tl
5245         { \exp_not:n { #3 } }
5246         \exp_not:n
5247         {
5248             {
5249                 \bool_if:NTF \l_@@_NiceTabular_bool
5250                 {

```

```

5251         \group_begin:
5252         \cs_set:Npn \arraystretch { 1 }
5253         \dim_set_eq:NN \extrarowheight \c_zero_dim
5254         #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).

```

5255         \bool_if:NT \g_@@_rotate_bool
5256         { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5257         \use:x
5258         {
5259             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5260             { @ { } \l_@@_hpos_of_block_tl @ { } }
5261         }
5262         #5
5263         \end { tabular }
5264         \group_end:
5265     }
5266 {
5267     \group_begin:
5268     \cs_set:Npn \arraystretch { 1 }
5269     \dim_set_eq:NN \extrarowheight \c_zero_dim
5270     #4
5271     \bool_if:NT \g_@@_rotate_bool
5272     { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5273     \c_math_toggle_token
5274     \use:x
5275     {
5276         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5277         { @ { } \l_@@_hpos_of_block_tl @ { } }
5278     }
5279     #5
5280     \end { array }
5281     \c_math_toggle_token
5282     \group_end:
5283 }
5284 }
5285 }
5286 }
5287 }

```

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).

```

5288 \keys_define:nn { NiceMatrix / Block / SecondPass }
5289 {
5290     fill .tl_set:N = \l_@@_fill_tl ,
5291     fill .value_required:n = true ,
5292     draw .tl_set:N = \l_@@_draw_tl ,
5293     draw .default:n = default ,
5294     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5295     rounded-corners .default:n = 4 pt ,
5296     color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5297     color .value_required:n = true ,
5298     borders .clist_set:N = \l_@@_borders_clist ,
5299     borders .value_required:n = true ,
5300     hvlines .bool_set:N = \l_@@_hvlines_block_bool ,
5301     hvlines .default:n = true ,
5302     line-width .dim_set:N = \l_@@_line_width_dim ,

```

```

5303   line-width .value_required:n = true ,
5304   l .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
5305   l .value_forbidden:n = true ,
5306   r .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
5307   r .value_forbidden:n = true ,
5308   c .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
5309   c .value_forbidden:n = true ,
5310   L .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l
5311             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5312   L .value_forbidden:n = true ,
5313   R .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r
5314             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5315   R .value_forbidden:n = true ,
5316   C .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c
5317             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5318   C .value_forbidden:n = true ,
5319   t .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl t ,
5320   t .value_forbidden:n = true ,
5321   b .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl b ,
5322   b .value_forbidden:n = true ,
5323   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
5324 }

```

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.

```

5325 \cs_new_protected:Npn \@@_draw_blocks:
5326 {
5327   \cs_set_eq:NN \ialign \@@_old_ialign:
5328   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
5329 }
5330 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5331 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5332   \int_zero_new:N \l_@@_last_row_int
5333   \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.

```

5334   \int_compare:nNnTF { #3 } > { 99 }
5335   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
5336   { \int_set:Nn \l_@@_last_row_int { #3 } }
5337   \int_compare:nNnTF { #4 } > { 99 }
5338   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
5339   { \int_set:Nn \l_@@_last_col_int { #4 } }
5340   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
5341   {
5342     \int_compare:nTF
5343     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
5344     {
5345       \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
5346       \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
5347       \group_begin:
5348       \globaldefs = 1
5349       \@@_msg_redirect_name:nn { columns~not~used } { none }
5350       \group_end:

```

```

5351     }
5352     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5353   }
5354   {
5355     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
5356     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5357     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
5358   }
5359 }
5360 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
5361 {

```

The sequence of the positions of the blocks will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

5362   \seq_gput_left:Nn \g_@@_pos_of_blocks_seq { { #1 } { #2 } { #3 } { #4 } }

```

The group is for the keys.

```

5363   \group_begin:
5364   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
5365   \tl_if_empty:NF \l_@@_draw_tl
5366   {
5367     \tl_gput_right:Nx \g_nicematrix_code_after_tl
5368     {
5369       \@@_stroke_block:nnn
5370       { \exp_not:n { #5 } }
5371       { #1 - #2 }
5372       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5373     }
5374     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
5375     { { #1 } { #2 } { #3 } { #4 } }
5376   }
5377   \bool_if:NT \l_@@_hvlines_block_bool
5378   {
5379     \tl_gput_right:Nx \g_nicematrix_code_after_tl
5380     {
5381       \@@_hvlines_block:nnn
5382       { \exp_not:n { #5 } }
5383       { #1 - #2 }
5384       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5385     }
5386   }
5387   \clist_if_empty:NF \l_@@_borders_clist
5388   {
5389     \tl_gput_right:Nx \g_nicematrix_code_after_tl
5390     {
5391       \@@_stroke_borders_block:nnn
5392       { \exp_not:n { #5 } }
5393       { #1 - #2 }
5394       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5395     }
5396   }
5397   \tl_if_empty:NF \l_@@_fill_tl
5398   {

```

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`.

```

5399     \exp_last_unbraced:NV \@@_extract_brackets \l_@@_fill_tl \q_stop
5400     \tl_gput_right:Nx \g_nicematrix_code_before_tl
5401     {
5402       \exp_not:N \roundedrectanglecolor
5403       [ \l_tmpa_tl ]
5404       { \exp_not:V \l_tmpb_tl }

```

```

5405         { #1 - #2 }
5406         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5407         { \dim_use:N \l_@@_rounded_corners_dim }
5408     }
5409 }

5410 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5411 {
5412     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5413     {
5414         \@@_actually_diagbox:nnnnnn
5415         { #1 }
5416         { #2 }
5417         { \int_use:N \l_@@_last_row_int }
5418         { \int_use:N \l_@@_last_col_int }
5419         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5420     }
5421 }

5422 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
5423 \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.

```

5424 \pgfpicture
5425     \pgfrememberpicturepositiononpagetrue
5426     \pgf@relevantforpicturesizefalse
5427     \@@_qpoint:n { row - #1 }
5428     \dim_set_eq:NN \l_tmpa_dim \pgf@y
5429     \@@_qpoint:n { col - #2 }
5430     \dim_set_eq:NN \l_tmpb_dim \pgf@x
5431     \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
5432     \dim_set_eq:NN \l_tmpc_dim \pgf@y
5433     \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5434     \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.

```

5435 \begin { pgfscope }
5436 \@@_pgf_rect_node:nnnnn
5437 { \@@_env: - #1 - #2 - block }
5438 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
5439 \end { pgfscope }

```

Now, we create the “short node” which, in general, will be used to put the label (that is to say the content of the node). However, if one the keys L, C or R is used (that information is provided by the boolean `\l_@@_hpos_of_block_cap_bool`), we don’t need to create that node since the normal node is used to put the label.

```

5440 \bool_if:NF \l_@@_hpos_of_block_cap_bool
5441 {
5442   \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.

```

5443   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5444   {

```

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.

```

5445     \cs_if_exist:cT
5446     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5447     {
5448       \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5449       {
5450         \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
5451         \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
5452       }
5453     }
5454   }

```

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.

```

5455   \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
5456   {
5457     \@@_qpoint:n { col - #2 }
5458     \dim_set_eq:NN \l_tmpb_dim \pgf@x
5459   }
5460   \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
5461   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5462   {
5463     \cs_if_exist:cT
5464     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5465     {
5466       \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5467       {
5468         \pgfpointanchor
5469         { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5470         { east }
5471         \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
5472       }
5473     }
5474   }
5475   \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
5476   {
5477     \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5478     \dim_set_eq:NN \l_tmpd_dim \pgf@x
5479   }
5480   \@@_pgf_rect_node:nnnnn
5481   { \@@_env: - #1 - #2 - block - short }
5482   \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpe_dim
5483 }

```

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.

```

5484 \bool_if:NT \l_@@_medium_nodes_bool
5485 {
5486   \@@_pgf_rect_node:nnn

```



```

5487 { \@@_env: - #1 - #2 - block - medium }
5488 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
5489 {
5490   \pgfpointanchor
5491   { \@@_env:
5492     - \int_use:N \l_@@_last_row_int
5493     - \int_use:N \l_@@_last_col_int - medium
5494   }
5495   { south-east }
5496 }
5497 }

```

Now, we will put the label of the block beginning with the case of a \Block of one row.

```

5498 \int_compare:nNnTF { #1 } = { #3 }
5499 {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

5500   \int_compare:nNnTF { #1 } = 0
5501   { \l_@@_code_for_first_row_tl }
5502   {
5503     \int_compare:nNnT { #1 } = \l_@@_last_row_int
5504     \l_@@_code_for_last_row_tl
5505   }

```

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.

```

5506   \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in \pgf@x) the x -value of the center of the block.

```

5507   \pgfpointanchor
5508   {
5509     \@@_env: - #1 - #2 - block
5510     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
5511   }
5512   {
5513     \str_case:Vn \l_@@_hpos_of_block_tl
5514     {
5515       c { center }
5516       l { west }
5517       r { east }
5518     }
5519   }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

5520   \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
5521   \pgfset { inner~sep = \c_zero_dim }
5522   \pgfnode
5523   { rectangle }
5524   {
5525     \str_case:Vn \l_@@_hpos_of_block_tl
5526     {
5527       c { base }
5528       l { base~west }
5529       r { base~east }
5530     }
5531   }
5532   { \box_use_drop:N \l_@@_cell_box } { } { }
5533 }

```

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).

```

5534 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

5535     \int_compare:nNnT { #2 } = 0
5536     { \tl_set:Nn \l_@@_hpos_of_block_tl r }
5537     \bool_if:nT \g_@@_last_col_found_bool
5538     {
5539         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5540         { \tl_set:Nn \l_@@_hpos_of_block_tl 1 }
5541     }
5542     \pgftransformshift
5543     {
5544         \pgfpointanchor
5545         {
5546             \@@_env: - #1 - #2 - block
5547             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
5548         }
5549         {
5550             \str_case:Vn \l_@@_hpos_of_block_tl
5551             {
5552                 c { center }
5553                 l { west }
5554                 r { east }
5555             }
5556         }
5557     }
5558     \pgfset { inner-sep = \c_zero_dim }
5559     \pgfnode
5560     { rectangle }
5561     {
5562         \str_case:Vn \l_@@_hpos_of_block_tl
5563         {
5564             c { center }
5565             l { west }
5566             r { east }
5567         }
5568     }
5569     { \box_use_drop:N \l_@@_cell_box } { } { }
5570 }
5571 \endpgfpicture
5572 \group_end:
5573 }

```

```

5574 \NewDocumentCommand \@@_extract_brackets { 0 { } }
5575 {
5576     \tl_set:Nn \l_tmpa_tl { #1 }
5577     \@@_store_in_tmpb_tl
5578 }
5579 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
5580 { \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).

```

5581 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
5582 {
5583     \group_begin:
5584     \tl_clear:N \l_@@_draw_tl
5585     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5586     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
5587     \pgfpicture
5588     \pgfrememberpicturerepositiononpagetrue
5589     \pgf@relevantforpicturesizefalse
5590     \tl_if_empty:NF \l_@@_draw_tl

```

```

5591 {
If the user has used the key color of the command \Block without value, the color fixed by
\arrayrulecolor is used.
5592 \str_if_eq:VnTF \l_@@_draw_tl { default }
5593 { \CT@arc@ }
5594 { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
5595 }
5596 \pgfsetcornersarced
5597 {
5598 \pgfpoint
5599 { \dim_use:N \l_@@_rounded_corners_dim }
5600 { \dim_use:N \l_@@_rounded_corners_dim }
5601 }
5602 \@@_cut_on_hyphen:w #2 \q_stop
5603 \bool_lazy_and:nnT
5604 { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
5605 { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
5606 {
5607 \@@_qpoint:n { row - \l_tmpa_tl }
5608 \dim_set:Nn \l_tmpb_dim { \pgf@y }
5609 \@@_qpoint:n { col - \l_tmpb_tl }
5610 \dim_set:Nn \l_tmpc_dim { \pgf@x }
5611 \@@_cut_on_hyphen:w #3 \q_stop
5612 \int_compare:nNnT \l_tmpa_tl > \c@iRow
5613 { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
5614 \int_compare:nNnT \l_tmpb_tl > \c@jCol
5615 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5616 \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
5617 \dim_set:Nn \l_tmpa_dim { \pgf@y }
5618 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
5619 \dim_set:Nn \l_tmpd_dim { \pgf@x }
5620 \pgfpathrectanglecorners
5621 { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5622 { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5623 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use \pgfusepathqstroke because of the key rounded-corners.

```

5624 \pgfusepath { stroke }
5625 }
5626 \endpgfpicture
5627 \group_end:
5628 }

```

Here is the set of keys for the command \@@_stroke_block:nnn.

```

5629 \keys_define:nn { NiceMatrix / BlockStroke }
5630 {
5631 color .tl_set:N = \l_@@_draw_tl ,
5632 draw .tl_set:N = \l_@@_draw_tl ,
5633 draw .default:n = default ,
5634 line-width .dim_set:N = \l_@@_line_width_dim ,
5635 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5636 rounded-corners .default:n = 4 pt
5637 }

```

The first argument of \@@_hvlines_block:nnn is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

5638 \cs_new_protected:Npn \@@_hvlines_block:nnn #1 #2 #3
5639 {
5640 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5641 \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5642 \@@_cut_on_hyphen:w #2 \q_stop
5643 \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl

```

```

5644 \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5645 \@@_cut_on_hyphen:w #3 \q_stop
5646 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5647 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5648 \pgfpicture
5649 \pgfrememberpicturepositiononpagetrue
5650 \pgf@relevantforpicturesizefalse
5651 \CT@arc@
5652 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

First, the vertical rules.

```

5653 \@@_qpoint:n { row - \l_tmpa_tl }
5654 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5655 \@@_qpoint:n { row - \l_tmpc_tl }
5656 \dim_set_eq:NN \l_tmpb_dim \pgf@y
5657 \int_step_inline:nnn \l_tmpd_tl \l_tmpb_tl
5658 {
5659   \@@_qpoint:n { col - ##1 }
5660   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpa_dim }
5661   \pgfpathlineto { \pgfpoint \pgf@x \l_tmpb_dim }
5662   \pgfusepathqstroke
5663 }

```

Now, the horizontal rules.

```

5664 \@@_qpoint:n { col - \l_tmpb_tl }
5665 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
5666 \@@_qpoint:n { col - \l_tmpd_tl }
5667 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \arrayrulewidth }
5668 \int_step_inline:nnn \l_tmpc_tl \l_tmpa_tl
5669 {
5670   \@@_qpoint:n { row - ##1 }
5671   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5672   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5673   \pgfusepathqstroke
5674 }
5675 \endpgfpicture
5676 }

```

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).

```

5677 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
5678 {
5679   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5680   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5681   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
5682   { \@@_error:n { borders~forbidden } }
5683   {
5684     \clist_map_inline:Nn \l_@@_borders_clist
5685     {
5686       \clist_if_in:nnF { top , bottom , left , right } { ##1 }
5687       { \@@_error:nn { bad-border } { ##1 } }
5688     }
5689     \@@_cut_on_hyphen:w #2 \q_stop
5690     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5691     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5692     \@@_cut_on_hyphen:w #3 \q_stop
5693     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5694     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5695     \pgfpicture
5696     \pgfrememberpicturepositiononpagetrue
5697     \pgf@relevantforpicturesizefalse
5698     \CT@arc@
5699     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

```

5700     \clist_if_in:NnT \l_@@_borders_clist { right }
5701     { \@@_stroke_vertical:n \l_tmpb_tl }
5702     \clist_if_in:NnT \l_@@_borders_clist { left }
5703     { \@@_stroke_vertical:n \l_tmpd_tl }
5704     \clist_if_in:NnT \l_@@_borders_clist { bottom }
5705     { \@@_stroke_horizontal:n \l_tmpa_tl }
5706     \clist_if_in:NnT \l_@@_borders_clist { top }
5707     { \@@_stroke_horizontal:n \l_tmpc_tl }
5708     \endpgfpicture
5709   }
5710 }

```

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).

```

5711 \cs_new_protected:Npn \@@_stroke_vertical:n #1
5712 {
5713   \@@_qpoint:n \l_tmpc_tl
5714   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
5715   \@@_qpoint:n \l_tmpa_tl
5716   \dim_set:Nn \l_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
5717   \@@_qpoint:n { #1 }
5718   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
5719   \pgfpathlineto { \pgfpoint \pgf@x \l_tmpc_dim }
5720   \pgfusepathqstroke
5721 }

```

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).

```

5722 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
5723 {
5724   \@@_qpoint:n \l_tmpd_tl
5725   \clist_if_in:NnTF \l_@@_borders_clist { left }
5726   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
5727   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
5728   \@@_qpoint:n \l_tmpb_tl
5729   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
5730   \@@_qpoint:n { #1 }
5731   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5732   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5733   \pgfusepathqstroke
5734 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

5735 \keys_define:nn { NiceMatrix / BlockBorders }
5736 {
5737   borders .clist_set:N = \l_@@_borders_clist ,
5738   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5739   rounded-corners .default:n = 4 pt ,
5740   line-width .dim_set:N = \l_@@_line_width_dim
5741 }

```

How to draw the dotted lines transparently

```

5742 \cs_set_protected:Npn \@@_renew_matrix:
5743 {
5744   \RenewDocumentEnvironment { pmatrix } { } {
5745     { \pNiceMatrix }
5746     { \endpNiceMatrix }
5747   }
5748   \RenewDocumentEnvironment { vmatrix } { } {
5749     { \vNiceMatrix }
5750     { \endvNiceMatrix }
5751   }
5752   \RenewDocumentEnvironment { Vmatrix } { } {

```

```

5751 { \VNiceMatrix }
5752 { \endVNiceMatrix }
5753 \RenewDocumentEnvironment { bmatrix } { }
5754 { \bNiceMatrix }
5755 { \endbNiceMatrix }
5756 \RenewDocumentEnvironment { Bmatrix } { }
5757 { \BNiceMatrix }
5758 { \endBNiceMatrix }
5759 }

```

Automatic arrays

```

5760 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
5761 {
5762   \int_set:Nn \l_@@_nb_rows_int { #1 }
5763   \int_set:Nn \l_@@_nb_cols_int { #2 }
5764 }

```

We will extract the potential keys l, r and c and pass the other keys to the environment {NiceArrayWithDelims}.

```

5765 \keys_define:nn { NiceMatrix / Auto }
5766 {
5767   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
5768   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
5769   c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
5770 }
5771 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
5772 {
5773   \int_zero_new:N \l_@@_nb_rows_int
5774   \int_zero_new:N \l_@@_nb_cols_int
5775   \@@_set_size:n #4 \q_stop

```

The group is for the protection of \l_@@_type_of_col_tl.

```

5776 \group_begin:
5777 \tl_set:Nn \l_@@_type_of_col_tl c
5778 \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
5779 \use:x
5780 {
5781   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
5782   { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
5783   [ \l_tmpa_tl ]
5784 }
5785 \int_compare:nNnT \l_@@_first_row_int = 0
5786 {
5787   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5788   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5789   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5790 }
5791 \prg_replicate:nn \l_@@_nb_rows_int
5792 {
5793   \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).

```

5794   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
5795   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5796 }
5797 \int_compare:nNnT \l_@@_last_row_int > { -2 }
5798 {
5799   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5800   \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5801   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\

```

```

5802     }
5803     \end { NiceArrayWithDelims }
5804     \group_end:
5805 }
5806 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
5807 {
5808     \cs_set_protected:cpn { #1 AutoNiceMatrix }
5809     {
5810         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
5811         \AutoNiceMatrixWithDelims { #2 } { #3 }
5812     }
5813 }
5814 \@@_define_com:nnn p ( )
5815 \@@_define_com:nnn b [ ]
5816 \@@_define_com:nnn v | |
5817 \@@_define_com:nnn V \| \|
5818 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

5819 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
5820 {
5821     \group_begin:
5822     \bool_set_true:N \l_@@_NiceArray_bool
5823     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
5824     \group_end:
5825 }

```

The redefinition of the command `\dotfill`

```

5826 \cs_set_eq:NN \@@_old_dotfill \dotfill
5827 \cs_new_protected:Npn \@@_dotfill:
5828 {

```

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}`).

```

5829     \@@_old_dotfill
5830     \bool_if:NT \l_@@_NiceTabular_bool
5831     { \group_insert_after:N \@@_dotfill_ii: }
5832     { \group_insert_after:N \@@_dotfill_i: }
5833 }
5834 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
5835 \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`.

```

5836 \cs_new_protected:Npn \@@_dotfill_iii:
5837 { \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`.

```

5838 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
5839 {
5840     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5841     {
5842         \@@_actually_diagbox:nnnnnn
5843         { \int_use:N \c_iRow }
5844         { \int_use:N \c_jCol }
5845         { \int_use:N \c_iRow }
5846         { \int_use:N \c_jCol }
5847         { \exp_not:n { #1 } }

```

```

5848     { \exp_not:n { #2 } }
5849   }

```

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.

```

5850   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
5851   {
5852     { \int_use:N \c@iRow }
5853     { \int_use:N \c@jCol }
5854     { \int_use:N \c@iRow }
5855     { \int_use:N \c@jCol }
5856   }
5857 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The two other are the elements to draw below and above the diagonal line.

```

5858 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
5859 {
5860   \pgfpicture
5861   \pgf@relevantforpicturesizefalse
5862   \pgfrememberpicturepositiononpagetrue
5863   \@@_qpoint:n { row - #1 }
5864   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5865   \@@_qpoint:n { col - #2 }
5866   \dim_set_eq:NN \l_tmpb_dim \pgf@x
5867   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5868   \@@_qpoint:n { row - \@@_succ:n { #3 } }
5869   \dim_set_eq:NN \l_tmpc_dim \pgf@y
5870   \@@_qpoint:n { col - \@@_succ:n { #4 } }
5871   \dim_set_eq:NN \l_tmpd_dim \pgf@x
5872   \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
5873   {

```

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.

```

5874     \CT@arc@
5875     \pgfsetroundcap
5876     \pgfusepathqstroke
5877   }
5878   \pgfset { inner-sep = 1 pt }
5879   \pgfscope
5880   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
5881   \pgfnode { rectangle } { south-west }
5882   { \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }
5883   \endpgfscope
5884   \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5885   \pgfnode { rectangle } { north-east }
5886   { \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
5887   \endpgfpicture
5888 }

```

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.

```

5889 \keys_define:nn { NiceMatrix }
5890 {
5891   CodeAfter / rules .inherit:n = NiceMatrix / rules ,

```



```

5892   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
5893 }
5894 \keys_define:nn { NiceMatrix / CodeAfter }
5895 {
5896   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
5897   sub-matrix .value_required:n = true ,
5898   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
5899   delimiters / color .value_required:n = true ,
5900   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
5901   rules .value_required:n = true ,
5902   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
5903 }

```

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. 107.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

5904 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_i:n }

```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_i:n` which do *not* begin with `\omit` (and thus, the user will be able to use `\CodeAfter` without error and without the need to prefix by `\omit`).

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

5905 \cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
5906 {
5907   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
5908   \@@_CodeAfter_ii:n
5909 }

```

We catch the argument of the command `\end` (in `#1`).

```

5910 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1
5911 {

```

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.

```

5912   \str_if_eq:eeTF \currenenvir { #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`.

```

5913   {
5914     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
5915     \@@_CodeAfter_i:n
5916   }
5917 }

```

The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_internal_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of columnn. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```

5918 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3

```

```

5919 {
5920   \pgfpicture
5921   \pgfrememberpicturepositiononpagetrue
5922   \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.

```

5923   \@@_qpoint:n { row - 1 }
5924   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5925   \@@_qpoint:n { row - \@@_succ:n \c@iRow }
5926   \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).

```

5927   \bool_if:nTF { #3 }
5928   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
5929   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
5930   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5931   {
5932     \cs_if_exist:cT
5933     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5934     {
5935       \pgfpointanchor
5936       { \@@_env: - ##1 - #2 }
5937       { \bool_if:nTF { #3 } { west } { east } }
5938       \dim_set:Nn \l_tmpa_dim
5939       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
5940     }
5941   }

```

Now we can put the delimiter with a node of PGF.

```

5942   \pgfset { inner~sep = \c_zero_dim }
5943   \dim_zero:N \nulldelimiterspace
5944   \pgftransformshift
5945   {
5946     \pgfpoint
5947     { \l_tmpa_dim
5948       { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
5949     }
5950   \pgfnode
5951   { rectangle }
5952   { \bool_if:nTF { #3 } { east } { west } }
5953   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

5954     \nullfont
5955     \c_math_toggle_token
5956     \tl_if_empty:NF \l_@@_delimiters_color_tl
5957     { \color { \l_@@_delimiters_color_tl } }
5958     \bool_if:nTF { #3 } { \left #1 } { \left . }
5959     \vcenter
5960     {
5961       \nullfont
5962       \hrule \@height
5963       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
5964       \@depth \c_zero_dim
5965       \@width \c_zero_dim
5966     }
5967     \bool_if:nTF { #3 } { \right . } { \right #1 }
5968     \c_math_toggle_token
5969   }
5970   { }
5971   { }
5972 \endpgfpicture
5973 }

```

The command `\SubMatrix`

```

5974 \keys_define:nn { NiceMatrix / sub-matrix }
5975 {
5976   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
5977   extra-height .value_required:n = true ,
5978   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
5979   left-xshift .value_required:n = true ,
5980   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
5981   right-xshift .value_required:n = true ,
5982   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
5983   xshift .value_required:n = true ,
5984   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
5985   delimiters / color .value_required:n = true ,
5986   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
5987   slim .default:n = true ,
5988   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
5989   hlines .default:n = all ,
5990   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
5991   vlines .default:n = all ,
5992   hvlines .meta:n = { hlines, vlines } ,
5993   hvlines .value_forbidden:n = true ,
5994 }
5995 \keys_define:nn { NiceMatrix }
5996 {
5997   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
5998   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
5999   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6000   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6001   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6002   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6003 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

6004 \keys_define:nn { NiceMatrix / SubMatrix }
6005 {
6006   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6007   hlines .default:n = all ,
6008   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6009   vlines .default:n = all ,
6010   hvlines .meta:n = { hlines, vlines } ,
6011   hvlines .value_forbidden:n = true ,
6012   name .code:n =
6013     \tl_if_empty:nTF { #1 }
6014     { \@@_error:n { Invalid-name-format } }
6015     {
6016       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6017       {
6018         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6019         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
6020         {
6021           \str_set:Nn \l_@@_submatrix_name_str { #1 }
6022           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6023         }
6024       }
6025       { \@@_error:n { Invalid-name-format } }
6026     } ,
6027   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6028   rules .value_required:n = true ,
6029   code .tl_set:N = \l_@@_code_tl ,
6030   code .value_required:n = true ,
6031   name .value_required:n = true ,
6032   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }

```

```

6033 }

6034 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
6035 {
6036   \peek_remove_spaces:n
6037   {
6038     \@@_cut_on_hyphen:w #3 \q_stop
6039     \tl_clear_new:N \l_tmpc_tl
6040     \tl_clear_new:N \l_tmpd_tl
6041     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6042     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6043     \@@_cut_on_hyphen:w #2 \q_stop
6044     \seq_gput_right:Nx \g_@@_submatrix_seq
6045     { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
6046     \tl_gput_right:Nn \g_@@_internal_code_after_tl
6047     { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6048   }
6049 }

```

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.

```

6050 \NewDocumentCommand \@@_SubMatrix { m m m m 0 { } }
6051 {
6052   \peek_remove_spaces:n
6053   { \@@_sub_matrix:nnnnn { #1 } { #2 } { #3 } { #4 } { #5 } }
6054 }

6055 \cs_new_protected:Npn \@@_sub_matrix:nnnnn #1 #2 #3 #4 #5
6056 {
6057   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

6058 \tl_clear_new:N \l_@@_first_i_tl
6059 \tl_clear_new:N \l_@@_first_j_tl
6060 \tl_clear_new:N \l_@@_last_i_tl
6061 \tl_clear_new:N \l_@@_last_j_tl

```

The command `\@@_cut_on_hyphen:w` cuts on the hyphen an argument of the form $i-j$. The value of i is stored in `\l_tmpa_tl` and the value of j is stored in `\l_tmpb_tl`.

```

6062 \@@_cut_on_hyphen:w #2 \q_stop
6063 \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
6064 \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
6065 \@@_cut_on_hyphen:w #3 \q_stop
6066 \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
6067 \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
6068 \bool_lazy_or:nnTF
6069 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6070 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6071 { \@@_error:n { SubMatrix~too~large } }
6072 {
6073   \str_clear_new:N \l_@@_submatrix_name_str
6074   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6075   \pgfpicture
6076   \pgfrememberpicturepositiononpagetrue
6077   \pgf@relevantforpicturesizefalse

```

```

6078 \pgfset { inner-sep = \c_zero_dim }
6079 \dim_set_eq:Nn \l_@@_x_initial_dim \c_max_dim
6080 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currification.

```

6081 \bool_if:NTF \l_@@_submatrix_slim_bool
6082 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6083 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6084 {
6085   \cs_if_exist:cT
6086   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6087   {
6088     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6089     \dim_set:Nn \l_@@_x_initial_dim
6090     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6091   }
6092   \cs_if_exist:cT
6093   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6094   {
6095     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6096     \dim_set:Nn \l_@@_x_final_dim
6097     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6098   }
6099 }
6100 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
6101 { \@@_error:nn { impossible~delimiter } { left } }
6102 {
6103   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
6104   { \@@_error:nn { impossible~delimiter } { right } }
6105   { \@@_sub_matrix_i:nn { #1 } { #4 } }
6106 }
6107 \endpgfpicture
6108 }
6109 \group_end:
6110 }

```

#1 is the left delimiter dans #2 is the right one.

```

6111 \cs_new_protected:Npn \@@_sub_matrix_i:nn #1 #2
6112 {
6113   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
6114   \dim_set:Nn \l_@@_y_initial_dim
6115   { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
6116   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
6117   \dim_set:Nn \l_@@_y_final_dim
6118   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
6119   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
6120   {
6121     \cs_if_exist:cT
6122     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
6123     {
6124       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
6125       \dim_set:Nn \l_@@_y_initial_dim
6126       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6127     }
6128     \cs_if_exist:cT
6129     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6130     {
6131       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6132       \dim_set:Nn \l_@@_y_final_dim
6133       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6134     }
6135   }
6136   \dim_set:Nn \l_tmpa_dim

```

```

6137     {
6138         \l_@@_y_initial_dim - \l_@@_y_final_dim +
6139         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6140     }
6141     \dim_set_eq:Nn \nulldelimiterspace \c_zero_dim

```

We will draw the rules in the `\SubMatrix`.

```

6142     \group_begin:
6143     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6144     \tl_if_empty:NF \l_@@_rules_color_tl
6145     { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6146     \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`.

```

6147     \seq_map_inline:Nn \g_@@_cols_vlism_seq
6148     {
6149         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6150         {
6151             \int_compare:nNnT
6152             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6153             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

6154                 \@@_qpoint:n { col - ##1 }
6155                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6156                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6157                 \pgfusepathqstroke
6158             }
6159         }
6160     }

```

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.

```

6161     \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6162     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6163     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6164     {
6165         \bool_lazy_and:nnTF
6166         { \int_compare_p:nNn { ##1 } > 0 }
6167         {
6168             \int_compare_p:nNn
6169             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6170         {
6171             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6172             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6173             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6174             \pgfusepathqstroke
6175         }
6176         { \@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
6177     }

```

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.

```

6178     \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6179     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6180     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
6181     {
6182         \bool_lazy_and:nnTF
6183         { \int_compare_p:nNn { ##1 } > 0 }
6184         {
6185             \int_compare_p:nNn

```

```

6186         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6187     {
6188         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect \l_tmpa_dim and \l_tmpb_dim.

```

6189     \group_begin:

```

We compute in \l_tmpa_dim the x -value of the left end of the rule.

```

6190         \dim_set:Nn \l_tmpa_dim
6191         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6192         \str_case:nn { #1 }
6193         {
6194             ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6195             [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
6196             \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6197         }
6198         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in \l_tmpb_dim the x -value of the right end of the rule.

```

6199         \dim_set:Nn \l_tmpb_dim
6200         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6201         \str_case:nn { #2 }
6202         {
6203             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6204             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
6205             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6206         }
6207         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6208         \pgfusepathqstroke
6209         \group_end:
6210     }
6211     { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
6212 }

```

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).

```

6213     \str_if_empty:NF \l_@@_submatrix_name_str
6214     {
6215         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
6216         \l_@@_x_initial_dim \l_@@_y_initial_dim
6217         \l_@@_x_final_dim \l_@@_y_final_dim
6218     }
6219     \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.

```

6220     \begin { pgfscope }
6221     \pgftransformshift
6222     {
6223         \pgfpoint
6224         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6225         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6226     }
6227     \str_if_empty:NTF \l_@@_submatrix_name_str
6228     { \@@_node_left:nn #1 { } }
6229     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
6230     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

6231     \pgftransformshift
6232     {
6233         \pgfpoint
6234         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }

```

```

6235         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6236     }
6237     \str_if_empty:NTF \l_@@_submatrix_name_str
6238     { \@@_node_right:nn #2 { } }
6239     {
6240         \@@_node_right:nn #2 { \@@_env: - \l_@@_submatrix_name_str - right }
6241     }
6242     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
6243     \flag_clear_new:n { nicematrix }
6244     \l_@@_code_tl
6245 }

```

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`.

```

6246 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

6247 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
6248 {
6249     \use:e
6250     { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
6251 }

```

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 -.

```

6252 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
6253 { #1 { \@@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

6254 \tl_const:Nn \c_@@_integers_alist_tl
6255 {
6256     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
6257     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
6258     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
6259     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
6260 }

```

```

6261 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
6262 {

```

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 of a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

6263     \tl_if_empty:nTF { #2 }
6264     {
6265         \str_case:nVTF { #1 } \c_@@_integers_alist_tl
6266         {
6267             \flag_raise:n { nicematrix }

```



```

6268         \int_if_even:nTF { \flag_height:n { nicematrix } }
6269         { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
6270         { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
6271     }
6272     { #1 }
6273 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

6274     { \@@_pgfpointanchor_iii:w { #1 } #2 }
6275 }

```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

6276 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
6277 {
6278     \str_case:nnF { #1 }
6279     {
6280         { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
6281         { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
6282     }

```

Now the case of a node of the form $i-j$.

```

6283     {
6284         \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
6285         - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
6286     }
6287 }

```

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`).

```

6288 \cs_new_protected:Npn \@@_node_left:nn #1 #2
6289 {
6290     \pgfnode
6291     { rectangle }
6292     { east }
6293     {
6294         \nullfont
6295         \c_math_toggle_token
6296         \tl_if_empty:NF \l_@@_delimiters_color_tl
6297         { \color { \l_@@_delimiters_color_tl } }
6298         \left #1
6299         \vcenter
6300         {
6301             \nullfont
6302             \hrule \@height \l_tmpa_dim
6303             \c_zero_dim
6304             \@width \c_zero_dim
6305         }
6306         \right .
6307         \c_math_toggle_token
6308     }
6309     { #2 }
6310     { }
6311 }

```

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`).

```

6312 \cs_new_protected:Npn \@@_node_right:nn #1 #2
6313 {
6314     \pgfnode
6315     { rectangle }

```

```

6316     { west }
6317     {
6318         \nullfont
6319         \c_math_toggle_token
6320         \tl_if_empty:NF \l_@@_delimiters_color_tl
6321         { \color { \l_@@_delimiters_color_tl } }
6322         \left .
6323         \vcenter
6324         {
6325             \nullfont
6326             \hrule \@height \l_tmpa_dim
6327                 \@depth \c_zero_dim
6328                 \@width \c_zero_dim
6329         }
6330         \right #1
6331         \c_math_toggle_token
6332     }
6333     { #2 }
6334     { }
6335 }

```

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.

```

6336 \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.

```

6337 \bool_new:N \c_@@_footnote_bool

6338 \@@_msg_new:nnn { Unknown-option-for-package }
6339 {
6340     The~key~'\l_keys_key_str'~is-unknown. \\
6341     If-you-go-on,~it~will~be~ignored. \\
6342     For-a-list-of~the~available~keys,~type~H~<return>.
6343 }
6344 {
6345     The~available~keys~are~(in~alphabetic~order):~
6346     define-L-C-R,~
6347     footnote,~
6348     footnotehyper,~
6349     renew-dots,~and
6350     renew-matrix.
6351 }

```

Maybe we will completely delete the key 'transparent' in a future version.

```

6352 \@@_msg_new:nn { Key~transparent }
6353 {
6354     The~key~'transparent'~is~now~obsolete~(because~it's~name~
6355     is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
6356     and~'renew-matrix'.~However,~you~can~go~on.
6357 }

```

```

6358 \@@_msg_new:nn { define-L-C-R }
6359 {
6360   You~have~used~the~key~'define-L-C-R'.~Please~note~that~this~key~
6361   will~probably~be~deleted~in~a~future~version~of~'nicematrix'.\\
6362   However,~you~can~go~on.
6363 }
6364 \keys_define:nn { NiceMatrix / Package }
6365 {
6366   define-L-C-R .code:n =
6367     \@@_error:n { define-L-C-R }
6368     \bool_set_true:N \c_@@_define_L_C_R_bool ,
6369   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
6370   renew-dots .value_forbidden:n = true ,
6371   renew-matrix .code:n = \@@_renew_matrix: ,
6372   renew-matrix .value_forbidden:n = true ,
6373   transparent .code:n =
6374     {
6375       \@@_renew_matrix:
6376       \bool_set_true:N \l_@@_renew_dots_bool
6377       \@@_error:n { Key~transparent }
6378     } ,
6379   transparent .value_forbidden:n = true,
6380   footnote .bool_set:N = \c_@@_footnote_bool ,
6381   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
6382   unknown .code:n = \@@_error:n { Unknown~option~for~package }
6383 }
6384 \ProcessKeysOptions { NiceMatrix / Package }

6385 \@@_msg_new:nn { footnote~with~footnotehyper~package }
6386 {
6387   You~can't~use~the~option~'footnote'~because~the~package~
6388   footnotehyper~has~already~been~loaded.~
6389   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
6390   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6391   of~the~package~footnotehyper.\\
6392   If~you~go~on,~the~package~footnote~won't~be~loaded.
6393 }
6394 \@@_msg_new:nn { footnotehyper~with~footnote~package }
6395 {
6396   You~can't~use~the~option~'footnotehyper'~because~the~package~
6397   footnote~has~already~been~loaded.~
6398   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
6399   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6400   of~the~package~footnote.\\
6401   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
6402 }

6403 \bool_if:NT \c_@@_footnote_bool
6404 {
6405   \@ifclassloaded { beamer }
6406     { \bool_set_false:N \c_@@_footnote_bool }
6407     {
6408       \@ifpackageloaded { footnotehyper }
6409         { \@@_error:n { footnote~with~footnotehyper~package } }
6410         { \usepackage { footnote } }
6411       }
6412     }
6413 \bool_if:NT \c_@@_footnotehyper_bool
6414 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

6415 \ifclassloaded { beamer }
6416 { \bool_set_false:N \c_@@_footnote_bool }
6417 {
6418   \ifpackageloaded { footnote }
6419     { \@@_error:n { footnotehyper~with~footnote-package } }
6420     { \usepackage { footnotehyper } }
6421   }
6422   \bool_set_true:N \c_@@_footnote_bool
6423 }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

Error messages of the package

The following message will be deleted when we will delete the key `except-corners` for the command `\arraycolor`.

```

6424 \@@_msg_new:nn { key except-corners }
6425 {
6426   The~key~'except-corners'~has-been-deleted~for~the~command~\token_to_str:N
6427   \arraycolor\ in~the~\token_to_str:N \CodeBefore.~You~should~instead~use~
6428   the~key~'corners'~in~your~\@@_full_name_env:.\
6429   If~you~go~on,~this~key~will~be~ignored.
6430 }

6431 \seq_new:N \c_@@_types_of_matrix_seq
6432 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
6433 {
6434   NiceMatrix ,
6435   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
6436 }
6437 \seq_set_map_x:NNn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
6438 { \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`.

```

6439 \cs_new_protected:Npn \@@_error_too_much_cols:
6440 {
6441   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
6442   {
6443     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
6444     { \@@_fatal:n { too~much~cols~for~matrix } }
6445     {
6446       \bool_if:NF \l_@@_last_col_without_value_bool
6447       { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
6448     }
6449   }
6450   { \@@_fatal:n { too~much~cols~for~array } }
6451 }

```

The following command must *not* be protected since it's used in an error message.

```

6452 \cs_new:Npn \@@_message_hdotsfor:
6453 {
6454   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
6455   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
6456 }

```

```

6457 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
6458 {
6459   You-try-to-use-more-columns-than-allowed-by-your-
6460   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-
6461   columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus-the-
6462   exterior-columns).~This-error-is-fatal.
6463 }
6464 \@@_msg_new:nn { too-much-cols-for-matrix }
6465 {
6466   You-try-to-use-more-columns-than-allowed-by-your-
6467   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal-
6468   number-of-columns-for-a-matrix-is-fixed-by-the-LaTeX-counter-
6469   'MaxMatrixCols'.~Its-actual-value-is~\int_use:N \c@MaxMatrixCols.~
6470   This-error-is-fatal.
6471 }

```

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`.

```

6472 \@@_msg_new:nn { too-much-cols-for-array }
6473 {
6474   You-try-to-use-more-columns-than-allowed-by-your-
6475   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
6476   \int_use:N \g_@@_static_num_of_col_int\
6477   ~ (plus-the-potential-exterior-ones).~
6478   This-error-is-fatal.
6479 }
6480 \@@_msg_new:nn { last-col-not-used }
6481 {
6482   The-key~'last-col'~is-in-force-but-you-have-not-used-that-last-column~
6483   in-your~\@@_full_name_env:~.~However,~you-can-go-on.
6484 }
6485 \@@_msg_new:nn { columns-not-used }
6486 {
6487   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
6488   \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
6489   You-can-go-on-but-the-columns-you-did-not-used-won't-be-created.
6490 }
6491 \@@_msg_new:nn { in-first-col }
6492 {
6493   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
6494   If-you-go-on,~this-command-will-be-ignored.
6495 }
6496 \@@_msg_new:nn { in-last-col }
6497 {
6498   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
6499   If-you-go-on,~this-command-will-be-ignored.
6500 }
6501 \@@_msg_new:nn { in-first-row }
6502 {
6503   You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
6504   If-you-go-on,~this-command-will-be-ignored.
6505 }
6506 \@@_msg_new:nn { in-last-row }
6507 {
6508   You-can't-use-the-command~#1 in-the-last-row~(exterior)~of-the-array.\\
6509   If-you-go-on,~this-command-will-be-ignored.
6510 }
6511 \@@_msg_new:nn { double-closing-delimiter }
6512 {
6513   You-can't-put-a-second-closing-delimiter~"#1"~just-after-a-first-closing-

```

```

6514     delimiter.~This-delimiter~will~be~ignored.
6515 }
6516 \@@_msg_new:nn { delimiter~after~opening }
6517 {
6518     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
6519     delimiter.~This-delimiter~will~be~ignored.
6520 }
6521 \@@_msg_new:nn { bad~option~for~line~style }
6522 {
6523     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
6524     is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
6525 }
6526 \@@_msg_new:nn { Unknown~key~for~xdots }
6527 {
6528     As~for~now,~there~is~only~three~key~available~here:~'color',~'line~style'~
6529     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
6530     this~key~will~be~ignored.
6531 }
6532 \@@_msg_new:nn { Unknown~key~for~rowcolors }
6533 {
6534     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
6535     (and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
6536     this~key~will~be~ignored.
6537 }
6538 \@@_msg_new:nn { ampersand~in~light~syntax }
6539 {
6540     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
6541     ~you~have~used~the~key~'light~syntax'.~This~error~is~fatal.
6542 }
6543 \@@_msg_new:nn { SubMatrix~too~large }
6544 {
6545     Your~command~\token_to_str:N \SubMatrix\
6546     can't~be~drawn~because~your~matrix~is~too~small.\\
6547     If~you~go~on,~this~command~will~be~ignored.
6548 }
6549 \@@_msg_new:nn { double~backslash~in~light~syntax }
6550 {
6551     You~can't~use~\token_to_str:N \\~to~separate~rows~because~you~have~used~
6552     the~key~'light~syntax'.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
6553     (set~by~the~key~'end~of~row').~This~error~is~fatal.
6554 }
6555 \@@_msg_new:nn { standard~cline~in~document }
6556 {
6557     The~key~'standard~cline'~is~available~only~in~the~preamble.\\
6558     If~you~go~on~this~command~will~be~ignored.
6559 }
6560 \@@_msg_new:nn { old~column~type }
6561 {
6562     The~column~type~'#1'~is~no~longer~defined~in~'nicematrix'.~
6563     Since~version~5.0,~you~have~to~use~'l',~'c'~and~'r'~instead~of~'L',~
6564     'C'~and~'R'.~You~can~also~use~the~key~'define~L~C~R'.\\
6565     This~error~is~fatal.
6566 }
6567 \@@_msg_new:nn { bad~value~for~baseline }
6568 {
6569     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
6570     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
6571     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
6572     If~you~go~on,~a~value~of~1~will~be~used.
6573 }

```

```

6574 \@@_msg_new:nn { Invalid-name-format }
6575 {
6576   You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
6577   \SubMatrix.\\
6578   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
6579   If~you~go~on,~this~key~will~be~ignored.
6580 }
6581 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
6582 {
6583   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
6584   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
6585   number~is~not~valid.~If~you~go~on,~it~will~be~ignored.
6586 }
6587 \@@_msg_new:nn { impossible-delimiter }
6588 {
6589   It's~impossible~to~draw~the~#1~delimiter~of~your~
6590   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
6591   in~that~column.
6592   \bool_if:NT \l_@@_submatrix_slim_bool
6593   { ~Maybe~you~should~try~without~the~key~'slim'. } \\
6594   If~you~go~on,~this~\token_to_str:N \SubMatrix\ will~be~ignored.
6595 }
6596 \@@_msg_new:nn { empty-environment }
6597 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }
6598 \@@_msg_new:nn { Delimiter-with-small }
6599 {
6600   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
6601   because~the~key~'small'~is~in~force.\\
6602   This~error~is~fatal.
6603 }
6604 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
6605 {
6606   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code~after'~
6607   can't~be~executed~because~a~cell~doesn't~exist.\\
6608   If~you~go~on~this~command~will~be~ignored.
6609 }
6610 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
6611 {
6612   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
6613   in~this~\@@_full_name_env:.\
6614   If~you~go~on,~this~key~will~be~ignored.\\
6615   For~a~list~of~the~names~already~used,~type~H~<return>.
6616 }
6617 {
6618   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
6619   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
6620 }
6621 \@@_msg_new:nn { r-or-l-with-preamble }
6622 {
6623   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
6624   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
6625   your~\@@_full_name_env:.\
6626   If~you~go~on,~this~key~will~be~ignored.
6627 }
6628 \@@_msg_new:nn { Hdotsfor-in-col-0 }
6629 {
6630   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
6631   the~array.~This~error~is~fatal.
6632 }
6633 \@@_msg_new:nn { bad-corner }

```

```

6634 {
6635     #1~is~an~incorrect~specification~for~a~corner~(in~the~keys~
6636     'corners'~and~'except-corners').~The~available~
6637     values~are:~NW,~SW,~NE~and~SE.\\
6638     If~you~go~on,~this~specification~of~corner~will~be~ignored.
6639 }

6640 \@@_msg_new:nn { bad-border }
6641 {
6642     #1~is~an~incorrect~specification~for~a~border~(in~the~key~
6643     'borders'~of~the~command~\token_to_str:N \Block).~The~available~
6644     values~are:~left,~right,~top~and~bottom.\\
6645     If~you~go~on,~this~specification~of~border~will~be~ignored.
6646 }

6647 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
6648 {
6649     In~the~\@@_full_name_env:,~you~must~use~the~key~
6650     'last-col'~without~value.\\
6651     However,~you~can~go~on~for~this~time~
6652     (the~value~'\l_keys_value_tl'~will~be~ignored).
6653 }

6654 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
6655 {
6656     In~\NiceMatrixoptions,~you~must~use~the~key~
6657     'last-col'~without~value.\\
6658     However,~you~can~go~on~for~this~time~
6659     (the~value~'\l_keys_value_tl'~will~be~ignored).
6660 }

6661 \@@_msg_new:nn { Block-too-large-1 }
6662 {
6663     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
6664     too~small~for~that~block. \\
6665 }

6666 \@@_msg_new:nn { Block-too-large-2 }
6667 {
6668     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
6669     \g_@@_static_num_of_col_int\
6670     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
6671     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
6672     (&)~at~the~end~of~the~first~row~of~your~
6673     \@@_full_name_env:.\\
6674     If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
6675 }

6676 \@@_msg_new:nn { unknown-column-type }
6677 {
6678     The~column~type~'#1'~in~your~\@@_full_name_env:\
6679     is~unknown. \\
6680     This~error~is~fatal.
6681 }

6682 \@@_msg_new:nn { tabularnote-forbidden }
6683 {
6684     You~can't~use~the~command~\token_to_str:N\tabularnote\
6685     ~in~a~\@@_full_name_env:..~This~command~is~available~only~in~
6686     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
6687     If~you~go~on,~this~command~will~be~ignored.
6688 }

6689 \@@_msg_new:nn { borders-forbidden }
6690 {
6691     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
6692     because~the~option~'rounded-corners'~
6693     is~in~force~with~a~non-zero~value.\\

```



```

6694     If~you~go~on,~this~key~will~be~ignored.
6695 }

6696 \@@_msg_new:nn { bottomrule~without~booktabs }
6697 {
6698     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
6699     loaded~'booktabs'.\\
6700     If~you~go~on,~this~key~will~be~ignored.
6701 }

6702 \@@_msg_new:nn { enumitem~not~loaded }
6703 {
6704     You~can't~use~the~command~\token_to_str:N\tabularnote\
6705     ~because~you~haven't~loaded~'enumitem'.\\
6706     If~you~go~on,~this~command~will~be~ignored.
6707 }

6708 \@@_msg_new:nn { Wrong~last~row }
6709 {
6710     You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
6711     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
6712     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
6713     last~row.~You~can~avoid~this~problem~by~using~'last-row'~
6714     without~value~(more~compilations~might~be~necessary).
6715 }

6716 \@@_msg_new:nn { Yet~in~env }
6717 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }

6718 \@@_msg_new:nn { Outside~math~mode }
6719 {
6720     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
6721     (and~not~in~\token_to_str:N \vcenter).\\
6722     This~error~is~fatal.
6723 }

6724 \@@_msg_new:nn { One~letter~allowed }
6725 {
6726     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
6727     If~you~go~on,~it~will~be~ignored.
6728 }

6729 \@@_msg_new:nnn { Unknown~key~for~Block }
6730 {
6731     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
6732     \Block.\\ If~you~go~on,~it~will~be~ignored. \\
6733     For~a~list~of~the~available~keys,~type~H~<return>.
6734 }
6735 {
6736     The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
6737     hvlines,~l,~line-width,~rounded-corners,~r~and~t.
6738 }

6739 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
6740 {
6741     The~key~'\l_keys_key_str'~is~unknown.\\
6742     If~you~go~on,~it~will~be~ignored. \\
6743     For~a~list~of~the~available~keys~in~\token_to_str:N
6744     \CodeAfter,~type~H~<return>.
6745 }
6746 {
6747     The~available~keys~are~(in~alphabetic~order):~
6748     delimiters/color,~
6749     rules~(with~the~subkeys~'color'~and~'width'),~
6750     sub-matrix~(several~subkeys)~
6751     and~xdots~(several~subkeys).~
6752     The~latter~is~for~the~command~\token_to_str:N \line.
6753 }

```

```

6754 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
6755 {
6756   The~key~'\l_keys_key_str'~is~unknown.\\
6757   If~you~go~on,~this~key~will~be~ignored. \\
6758   For~a~list~of~the~available~keys~in~\token_to_str:N
6759   \SubMatrix,~type~H~<return>.
6760 }
6761 {
6762   The~available~keys~are~(in~alphabetic~order):~
6763   'delimiters/color',~
6764   'extra-height',~
6765   'hlines',~
6766   'hvlines',~
6767   'left-xshift',~
6768   'name',~
6769   'right-xshift',~
6770   'rules'~(with~the~subkeys~'color'~and~'width'),~
6771   'slim',~
6772   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
6773   and~'right-xshift').\\
6774 }
6775 \@@_msg_new:nnn { Unknown~key~for~notes }
6776 {
6777   The~key~'\l_keys_key_str'~is~unknown.\\
6778   If~you~go~on,~it~will~be~ignored. \\
6779   For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
6780 }
6781 {
6782   The~available~keys~are~(in~alphabetic~order):~
6783   bottomrule,~
6784   code-after,~
6785   code-before,~
6786   enumitem-keys,~
6787   enumitem-keys-para,~
6788   para,~
6789   label-in-list,~
6790   label-in-tabular~and~
6791   style.
6792 }
6793 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
6794 {
6795   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
6796   \token_to_str:N \NiceMatrixOptions. \\
6797   If~you~go~on,~it~will~be~ignored. \\
6798   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6799 }
6800 {
6801   The~available~keys~are~(in~alphabetic~order):~
6802   allow-duplicate-names,~
6803   cell-space-bottom-limit,~
6804   cell-space-limits,~
6805   cell-space-top-limit,~
6806   code-for-first-col,~
6807   code-for-first-row,~
6808   code-for-last-col,~
6809   code-for-last-row,~
6810   corners,~
6811   create-extra-nodes,~
6812   create-medium-nodes,~
6813   create-large-nodes,~
6814   delimiters~(several~subkeys),~
6815   end-of-row,~
6816   first-col,~

```

```

6817     first-row,~
6818     hlines,~
6819     hvlines,~
6820     last-col,~
6821     last-row,~
6822     left-margin,~
6823     letter-for-dotted-lines,~
6824     light-syntax,~
6825     notes~(several~subkeys),~
6826     nullify-dots,~
6827     renew-dots,~
6828     renew-matrix,~
6829     right-margin,~
6830     rules~(with~the~subkeys~'color'~and~'width'),~
6831     small,~
6832     sub-matrix~(several~subkeys),
6833     vlines,~
6834     xdots~(several~subkeys).
6835 }

6836 \@@_msg_new:nnn { Unknown~option~for~NiceArray }
6837 {
6838     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
6839     \{NiceArray\}. \\
6840     If~you~go~on,~it~will~be~ignored. \\
6841     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6842 }
6843 {
6844     The~available~keys~are~(in~alphabetic~order):~
6845     b,~
6846     baseline,~
6847     c,~
6848     cell-space-bottom-limit,~
6849     cell-space-limits,~
6850     cell-space-top-limit,~
6851     code-after,~
6852     code-for-first-col,~
6853     code-for-first-row,~
6854     code-for-last-col,~
6855     code-for-last-row,~
6856     colortbl-like,~
6857     columns-width,~
6858     corners,~
6859     create-extra-nodes,~
6860     create-medium-nodes,~
6861     create-large-nodes,~
6862     delimiters/color,~
6863     extra-left-margin,~
6864     extra-right-margin,~
6865     first-col,~
6866     first-row,~
6867     hlines,~
6868     hvlines,~
6869     last-col,~
6870     last-row,~
6871     left-margin,~
6872     light-syntax,~
6873     name,~
6874     notes/bottomrule,~
6875     notes/para,~
6876     nullify-dots,~
6877     renew-dots,~
6878     right-margin,~
6879     rules~(with~the~subkeys~'color'~and~'width'),~

```

```

6880     small,~
6881     t,~
6882     tabularnote,~
6883     vlines,~
6884     xdots/color,~
6885     xdots/shorten~and~
6886     xdots/line-style.
6887 }

```

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).

```

6888 \@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
6889 {
6890   The~key~'\l_keys_key_str'~is~unknown~for~the~
6891   \@@_full_name_env:. \\\
6892   If~you~go~on,~it~will~be~ignored. \\\
6893   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6894 }
6895 {
6896   The~available~keys~are~(in~alphabetic~order):~
6897   b,~
6898   baseline,~
6899   c,~
6900   cell-space-bottom-limit,~
6901   cell-space-limits,~
6902   cell-space-top-limit,~
6903   code-after,~
6904   code-for-first-col,~
6905   code-for-first-row,~
6906   code-for-last-col,~
6907   code-for-last-row,~
6908   colortbl-like,~
6909   columns-width,~
6910   corners,~
6911   create-extra-nodes,~
6912   create-medium-nodes,~
6913   create-large-nodes,~
6914   delimiters~(several~subkeys),~
6915   extra-left-margin,~
6916   extra-right-margin,~
6917   first-col,~
6918   first-row,~
6919   hlines,~
6920   hvlines,~
6921   l,~
6922   last-col,~
6923   last-row,~
6924   left-margin,~
6925   light-syntax,~
6926   name,~
6927   nullify-dots,~
6928   r,~
6929   renew-dots,~
6930   right-margin,~
6931   rules~(with~the~subkeys~'color'~and~'width'),~
6932   small,~
6933   t,~
6934   vlines,~
6935   xdots/color,~
6936   xdots/shorten~and~
6937   xdots/line-style.
6938 }
6939 \@@_msg_new:nnn { Unknown~option~for~NiceTabular }

```

```

6940 {
6941   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
6942   \{NiceTabular\}. \\
6943   If~you~go~on,~it~will~be~ignored. \\
6944   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
6945 }
6946 {
6947   The~available~keys~are~(in~alphabetic~order):~
6948   b,~
6949   baseline,~
6950   c,~
6951   cell-space-bottom-limit,~
6952   cell-space-limits,~
6953   cell-space-top-limit,~
6954   code-after,~
6955   code-for-first-col,~
6956   code-for-first-row,~
6957   code-for-last-col,~
6958   code-for-last-row,~
6959   colortbl-like,~
6960   columns-width,~
6961   corners,~
6962   create-extra-nodes,~
6963   create-medium-nodes,~
6964   create-large-nodes,~
6965   extra-left-margin,~
6966   extra-right-margin,~
6967   first-col,~
6968   first-row,~
6969   hlines,~
6970   hvlines,~
6971   last-col,~
6972   last-row,~
6973   left-margin,~
6974   light-syntax,~
6975   name,~
6976   notes/bottomrule,~
6977   notes/para,~
6978   nullify-dots,~
6979   renew-dots,~
6980   right-margin,~
6981   rules~(with~the~subkeys~'color'~and~'width'),~
6982   t,~
6983   tabularnote,~
6984   vl原因,~
6985   xdots/color,~
6986   xdots/shorten~and~
6987   xdots/line-style.
6988 }
6989 \@@_msg_new:nnn { Duplicate~name }
6990 {
6991   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
6992   the~same~environment~name~twice.~You~can~go~on,~but,~
6993   maybe,~you~will~have~incorrect~results~especially~
6994   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
6995   message~again,~use~the~key~'allow-duplicate-names'~in~
6996   '\token_to_str:N \NiceMatrixOptions'.\\
6997   For~a~list~of~the~names~already~used,~type~H~<return>. \\
6998 }
6999 {
7000   The~names~already~defined~in~this~document~are:~
7001   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
7002 }

```

```

7003 \@@_msg_new:nn { Option~auto~for~columns~width }
7004 {
7005     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
7006     If~you~go~on,~the~key~will~be~ignored.
7007 }

```

18 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 StackExchange⁶³, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁶⁴

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\dashline` 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`.

⁶³cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁶⁴Before 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 2.3 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 StackExchange⁶⁵, 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`.

⁶⁵cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.

New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programming for the command `\Block` when the block has only one row. With this programming, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with `\multicolumn`. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands `\NiceMatrixLastEnv` and `\OnlyMainNiceMatrix`.

New options `create-medium-nodes` and `create-large-nodes`.

Changes between version 3.9 and 3.10

New option `light-syntax` (and `end-of-row`).

New option `dotted-lines-margin` for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

Correction of a bug linked to `first-row` and `last-row`.

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -block and, if the creation of the “medium nodes” is required, a node $i-j$ -block-medium is created.

If the user tries to use more columns than allowed by its environment, an error is raised by `nicematrix` (instead of a low-level error).

The package must be loaded with the option `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the “last row”.

The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customisation of the dotted lines.

In the environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, etc.), it's possible to use the options `l` (=L) or `r` (=R) to specify the type of the columns.

The starred versions of the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots` and `\Iddots` are deprecated since the version 3.1 of `nicematrix`. Now, one should load `nicematrix` with the option `starred-commands` to avoid an error at the compilation.

The code of `nicematrix` no longer uses Tikz but only PGF. By default, Tikz is *not* loaded by `nicematrix`.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on [stackoverflow](#)).

Better error messages when the user uses `&` or `\\` when `light-syntax` is in force.

Changes between versions 3.14 and 3.15

It's possible to put labels on the dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and the command `\line` in the code-after with the tokens `_` and `^`.

The option `baseline` is now available in all the environments of `nicematrix`. Before, it was available only in `{NiceArray}`.

New keyword `\CodeAfter` (in the environments of `nicematrix`).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`

Commands to color cells, rows and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`

New command `\diagbox`

The key `hvhline` 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 `hvhlines-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 (that key will probably be deleted in a future version of `nicematrix`).

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).

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

@@ commands:

`\@@_Block:` ~~1230~~, ~~509~~212

\@@_Block_i	5099, 5100, 5104	\@@_add_to_colors_seq:nn	3965, 3977, 4000, 4015, 4030, 4039
\@@_Block_ii:nnnnn	5104, 5105	\@@_adjust_pos_of_blocks_seq: ..	2707, 2763
\@@_Block_iv:nnnnn	5135, 5139	\@@_adjust_pos_of_blocks_seq:i:nnnn ..	2766, 2768
\@@_Block_iv:nnnnnn	5328, 5330	\@@_adjust_size_box:	954, 980, 1805, 2047, 2509, 2554
\@@_Block_v:nnnnn	5136, 5240	\@@_adjust_to_submatrix:nn	3048, 3151, 3190, 3271, 3346, 3398
\@@_Block_v:nnnnnn	5357, 5360	\@@_adjust_to_submatrix:nnnnnn ..	3055, 3057
\@@_Cdots	1153, 1220, 3647	\@@_after_array:	1583, 2640
\g_@@_Cdots_lines_tl	1247, 2800	\g_@@_after_col_zero_bool	305, 1120, 2486, 3745
\@@_Cell:	198, 217, 876, 1724, 1774, 1796, 1996, 2017, 2038, 2607	\@@_analyze_end:Nn	2275, 2320
\@@_CodeAfter:	1234, 5904	\l_@@_argspec_tl	3629, 3630, 3631, 3647, 3663, 3679, 3703, 3759, 3760, 3761, 3836, 3837, 3838, 3914, 3915, 3916
\@@_CodeAfter_i:n	878, 2485, 2530, 5904, 5905, 5915	\@@_array:	1074, 2276, 2303
\@@_CodeAfter_ii:n	5908, 5910	\@@_arraycolor	1356, 4076
\@@_CodeAfter_keys:	2739, 2761	\c_@@_arydshln_loaded_bool ...	24, 31, 1710
\@@_CodeBefore:w	1387, 1389	\l_@@_auto_columns_width_bool	505, 652, 2387, 2391, 4849
\@@_CodeBefore_keys:	1367, 1384	\g_@@_aux_found_bool	1264, 1269, 1391, 1460, 1463
\@@_Ddots	1155, 1222, 3679	\g_@@_aux_tl	280, 1466, 1589, 2649, 2664, 2673, 2748, 4670
\g_@@_Ddots_lines_tl	1250, 2798	\l_@@_bar_at_end_of_pream_bool	1636, 1768, 2453
\g_@@_HVDotsfor_lines_tl	1252, 2796, 3763, 3840, 6454	\l_@@_baseline_tl	496, 497, 645, 646, 647, 648, 1087, 1516, 2088, 2100, 2105, 2107, 2112, 2117, 2199, 2200, 2204, 2209, 2211, 2216
\@@_Hdotsfor:	1158, 1227, 3739	\@@_begin_of_NiceMatrix:nn	2596, 2617
\@@_Hdotsfor:nnnn	3765, 3778	\@@_begin_of_row:	881, 904, 1179, 2487
\@@_Hdotsfor_i	3748, 3754, 3761	\l_@@_block_auto_columns_width_bool ..	1453, 2392, 4842, 4847, 4857, 4867
\@@_Hline:	1225, 4587	\g_@@_block_box_int	346, 1433, 5141, 5155, 5157, 5193, 5205, 5217, 5226, 5236
\@@_Hline_i:n	4587, 4588, 4594	\g_@@_blocks_dp_dim	272, 962, 965, 966, 5220, 5223
\@@_Hline_ii:nn	4591, 4594	\g_@@_blocks_ht_dim	271, 968, 971, 972, 5211, 5214
\@@_Hline_iii:n	4592, 4595	\g_@@_blocks_seq	319, 1455, 2138, 5230, 5242, 5328
\@@_Hspace:	1226, 3733	\g_@@_blocks_wd_dim	270, 956, 959, 960, 5199, 5202
\@@_Iddots	1156, 1223, 3703	\c_@@_booktabs_loaded_bool	25, 34, 1168, 2170
\g_@@_Iddots_lines_tl	1251, 2799	\l_@@_borders_clist	335, 5298, 5387, 5684, 5700, 5702, 5704, 5706, 5725, 5737
\@@_Ldots	1152, 1157, 1219, 3631	\@@_cartesian_path:	4009, 4024, 4138, 4150, 4243
\g_@@_Ldots_lines_tl	1248, 2801	\@@_cartesian_path:n	4054, 4185, 4243
\l_@@_Matrix_bool	278, 1574, 1597, 2598	\l_@@_cell_box	882, 928, 930, 936, 942, 945, 949, 958, 959, 964, 965, 970, 971, 981, 982, 983, 984, 986, 989, 993, 995, 1014, 1029, 1031, 1038, 1039, 1052, 1170, 1277, 1279, 1795, 1806, 2037, 2048, 2488, 2512, 2515, 2517, 2534, 2557, 2561, 5422, 5532, 5569, 5837
\l_@@_NiceArray_bool	275, 402, 1293, 1472, 1514, 1635, 1642, 1654, 2449, 2574, 4443, 4447, 4573, 4577, 4795, 4802, 5822	\l_@@_cell_space_bottom_limit_dim ...	485, 558, 984
\g_@@_NiceMatrixBlock_int	264, 4854, 4859, 4862, 4873	\l_@@_cell_space_top_limit_dim	484, 556, 982
\l_@@_NiceTabular_bool	168, 276, 883, 1079, 1366, 1369, 1440, 1547, 1551, 1643, 1655, 2450, 2627, 2636, 2738, 2741, 5167, 5249, 5830	\l_@@_cell_type_tl	268, 269, 1724, 1797, 1996, 2039, 5119, 5121
\@@_NotEmpty:	1236, 2621	\@@_cellcolor	1351, 4056, 4068, 4069
\@@_OnlyMainNiceMatrix:n	1232, 4303		
\@@_OnlyMainNiceMatrix_i:n	4306, 4313, 4316		
\@@_SubMatrix	2720, 6050		
\@@_SubMatrix_in_code_before	1359, 6034		
\@@_Vdots	1154, 1221, 3663		
\g_@@_Vdots_lines_tl	1249, 2797		
\@@_Vdotsfor:	1228, 3838		
\@@_Vdotsfor:nnnn	3842, 3853		
\@@_W:	1600, 1682, 1983		
\@@_actually_color:	1368, 3978		
\@@_actually_diagbox:nnnnnn	5146, 5414, 5842, 5858		
\@@_actually_draw_Cdots:	3203, 3207		
\@@_actually_draw_Ddots:	3353, 3357		
\@@_actually_draw_Iddots:	3405, 3409		
\@@_actually_draw_Ldots: ..	3164, 3168, 3829		
\@@_actually_draw_Vdots: ..	3285, 3289, 3904		
\@@_adapt_S_column:	229, 234, 247, 1439		

\@@_cellcolor_tabular	1162, 4268	\@@_create_row_node:	1090, 1123, 1169
\g_@@_cells_seq	2314, 2315, 2316, 2318	\@@_cut_on_hyphen:w	3991, 4046, 4051, 4111, 4198,
\@@_chessboardcolors	1358, 4061	4199, 4221, 4222, 4252, 4253, 5602, 5611,	
\@@_cline	147, 1218	5642, 5645, 5689, 5692, 6038, 6043, 6062, 6065	
\@@_cline_i:nn	148, 149, 161, 164	\g_@@_ddots_int	2686, 3377, 3378
\@@_cline_i:w	149, 150	\@@_def_env:nnn	2580, 2591, 2592, 2593, 2594, 2595
\l_@@_code_before_bool	309, 642, 669, 1094, 1283, 1315, 1469,	\@@_define_L_C_R:	252, 1289
2334, 2351, 2369, 2400, 2426, 2461, 2754, 2756		\c_@@_define_L_C_R_bool ...	251, 1289, 6368
\g_@@_code_before_tl .	1459, 1467, 1470, 2750	\@@_define_com:nnn	5806, 5814, 5815, 5816, 5817, 5818
\l_@@_code_before_tl	308, 641, 1314, 1367, 1470	\@@_delimiter:nnn	1839, 1860, 1868, 1881, 1887, 1893, 5918
\l_@@_code_for_first_col_tl	575, 2499	\l_@@_delimiters_color_tl	518, 728, 1380, 1539, 1540, 1557, 1558,
\l_@@_code_for_first_row_tl .	579, 892, 5501	5898, 5956, 5957, 5984, 6296, 6297, 6320, 6321	
\l_@@_code_for_last_col_tl	577, 2543	\l_@@_delimiters_max_width_bool	519, 726, 1562
\l_@@_code_for_last_row_tl .	581, 899, 5504	\g_@@_delta_x_one_dim	2688, 3380, 3390
\l_@@_code_tl	300, 6029, 6244	\g_@@_delta_x_two_dim	2690, 3428, 3438
\l_@@_col_max_int	330, 2915, 2926, 2994, 3053, 3070	\g_@@_delta_y_one_dim	2689, 3382, 3390
\l_@@_col_min_int	329, 2920, 2983, 2988, 3051, 3068	\g_@@_delta_y_two_dim	2691, 3430, 3438
\g_@@_col_total_int 267, 997, 1243, 1323,		\@@_diagbox:nn	1235, 5838
1334, 1404, 1500, 1964, 1965, 2419, 2420,		\@@_dotfill:	5827
2464, 2468, 2473, 2474, 2533, 2644, 2646,		\@@_dotfill_i:	5832, 5834
2658, 2825, 3238, 3256, 3316, 3899, 4293,		\@@_dotfill_ii:	5831, 5834, 5835
4900, 4910, 4944, 5031, 5340, 5539, 6070, 6119		\@@_dotfill_iii:	5835, 5836
\l_@@_color_tl	337, 5091, 5159, 5161	\@@_double_int_eval:n	3910, 3924, 3925
\g_@@_colors_seq 1363, 3968, 3972, 3973, 3982		\g_@@_dp_ante_last_row_dim	907, 1203
\@@_colortbl_like:	1160, 1237	\g_@@_dp_last_row_dim	907, 908, 1206, 1207, 1278, 1279, 1533
\l_@@_colortbl_like_bool 482, 668, 1237, 1624		\g_@@_dp_row_zero_dim	927, 928, 1197, 1198, 1526, 2193, 2232
\c_@@_colortbl_loaded_bool .	100, 104, 1187	\@@_draw_Cdots:nnn	3188
\l_@@_cols_tl	4008, 4023, 4053, 4087, 4095, 4096, 4191, 4194	\@@_draw_Ddots:nnn	3344
\g_@@_cols_vlism_seq ..	288, 1619, 1701, 6147	\@@_draw_Iddots:nnn	3396
\@@_columncolor	1357, 4011	\@@_draw_Ldots:nnn	3149
\@@_columncolor:n	4017, 4020	\@@_draw_Vdots:nnn	3269
\@@_columncolor_preamble	1164, 4291	\@@_draw_blocks:	2138, 5325
\c_@@_columncolor_regex	59, 1627	\@@_draw_dotted_lines:	2705, 2785
\l_@@_columns_width_dim	265, 653, 795, 2388, 2394, 4855, 4861	\@@_draw_dotted_lines_i:	2788, 2792
\g_@@_com_or_env_str	292, 295	\l_@@_draw_first_bool .	344, 3694, 3718, 3729
\@@_computations_for_large_nodes: ...	4971, 4984, 4989	\@@_draw_hlines:	2718, 4569
\@@_computations_for_medium_nodes: ..	4891, 4960, 4970, 4981	\@@_draw_line:	3186,
\@@_compute_a_corner:nnnnnn	4658, 4660, 4662, 4664, 4678	3231, 3342, 3394, 3442, 3444, 3963, 4810, 4840	
\@@_compute_corners:	2706, 4650	\@@_draw_line_ii:nn	3943, 3947
\@@_construct_preamble:	1290, 1594	\@@_draw_line_iii:nn	3950, 3954
\l_@@_corners_cells_seq	323, 2712, 4188, 4228, 4377, 4383, 4390,	\@@_draw_non_standard_dotted_line: ..	3450, 3452
4509, 4515, 4522, 4652, 4668, 4672, 4673, 4734		\@@_draw_non_standard_dotted_line:n ..	3455, 3458
\l_@@_corners_clist	501, 630, 635, 4343, 4476, 4653	\@@_draw_non_standard_dotted_line:nnn	3460, 3465, 3479
\@@_create_col_nodes:	2279, 2307, 2326	\@@_draw_standard_dotted_line: .	3449, 3480
\@@_create_diag_nodes: ...	1340, 2683, 2867	\@@_draw_standard_dotted_line_i: 3543, 3547	
\@@_create_extra_nodes:	1422, 2137, 2859, 4881	\l_@@_draw_tl	334, 5292,
\@@_create_large_nodes:	4889, 4965	5296, 5365, 5584, 5590, 5592, 5594, 5631, 5632	
\@@_create_medium_and_large_nodes: ..	4886, 4976	\@@_draw_vlines:	2719, 4439
\@@_create_medium_nodes:	4887, 4955	\g_@@_empty_cell_bool	316, 988, 998,
\@@_create_nodes: 4962, 4973, 4983, 4986, 5027		2522, 2569, 3645, 3661, 3677, 3701, 3724, 3735	
		\@@_end_Cell:	203, 219,
		975, 1726, 1784, 1801, 1998, 2027, 2043, 2607	

\l_@@_end_of_row_tl	515, 516, 569, 2299, 2300, 6552	\l_@@_final_open_bool	2698, 2909, 2913, 2916, 2923, 2929, 2933, 2949, 3177, 3212, 3217, 3228, 3292, 3302, 3307, 3328, 3367, 3417, 3551, 3566, 3597, 3598, 3781, 3805, 3817, 3856, 3880, 3892, 3933, 4778, 4815
\c_@@_endpgfortikzpicture_tl	43, 47, 2789, 3951, 4781	\@@_find_extremities_of_line:nnnn	2897, 3154, 3193, 3274, 3349, 3401
\c_@@_enumitem_loaded_bool	26, 37, 375, 696, 701, 712, 717	\l_@@_first_col_int	135, 148, 356, 357, 571, 850, 881, 1334, 1404, 1509, 1637, 2329, 2349, 2656, 2825, 3238, 3256, 3743, 4212, 4305, 4900, 4910, 4944, 4992, 5031, 5787, 5793, 5799, 6119
\@@_env: ..	259, 263, 913, 919, 1015, 1021, 1035, 1043, 1099, 1105, 1111, 1182, 1324, 1325, 1330, 1331, 1336, 1337, 1348, 1401, 1402, 1407, 1410, 1413, 1416, 2335, 2338, 2340, 2356, 2362, 2365, 2374, 2380, 2383, 2405, 2411, 2414, 2431, 2437, 2443, 2464, 2468, 2474, 2729, 2827, 2831, 2837, 2844, 2850, 2884, 2891, 2893, 2894, 2955, 3023, 3087, 3098, 3111, 3114, 3133, 3136, 3241, 3244, 3259, 3262, 3792, 3810, 3867, 3885, 3936, 3938, 3957, 3960, 4689, 4708, 4726, 4913, 4915, 4923, 5034, 5043, 5061, 5437, 5446, 5450, 5464, 5469, 5481, 5487, 5488, 5491, 5509, 5546, 5933, 5936, 6086, 6088, 6093, 6095, 6122, 6124, 6129, 6131, 6229, 6240	\l_@@_first_i_tl	6058, 6063, 6082, 6113, 6122, 6124, 6179, 6186, 6188, 6269, 6280, 6284
\g_@@_env_int	258, 259, 261, 1452, 1461, 1464, 1588, 5070	\l_@@_first_j_tl	6059, 6064, 6086, 6088, 6149, 6162, 6169, 6171, 6270, 6281, 6285
\@@_error:n ...	12, 378, 403, 528, 538, 591, 722, 778, 788, 794, 803, 811, 829, 836, 844, 845, 846, 852, 857, 858, 859, 870, 872, 873, 874, 1382, 1495, 1505, 1577, 2122, 2175, 2221, 4075, 4090, 5323, 5682, 5902, 6014, 6025, 6032, 6071, 6367, 6377, 6382, 6409, 6419	\l_@@_first_row_int	354, 355, 572, 854, 1241, 1328, 1399, 1524, 2119, 2190, 2218, 2229, 2653, 2823, 3108, 3130, 4893, 4907, 4934, 4991, 5029, 5443, 5461, 5785, 5930, 6083, 6570
\@@_error:nn	13, 660, 1690, 1691, 1692, 1842, 1888, 1894, 1985, 1986, 1987, 3634, 3637, 3650, 3653, 3666, 3669, 3683, 3684, 3689, 3690, 3707, 3708, 3713, 3714, 4666, 5687, 6019, 6101, 6104	\c_@@_footnote_bool	1428, 1592, 6337, 6380, 6403, 6406, 6416, 6422
\@@_error:nnn	14, 3941, 6176, 6211	\c_@@_footnotehyper_bool	6336, 6381, 6413
\@@_error_too_much_cols:	1664, 6439	\@@_full_name_env:	293, 6428, 6460, 6467, 6475, 6483, 6487, 6584, 6597, 6600, 6613, 6618, 6623, 6625, 6649, 6668, 6673, 6678, 6685, 6711, 6720, 6891
\@@_everycr:	1116, 1192, 1195	\@@_hdottedline:	1224, 4763
\@@_everycr_i:	1116, 1117	\@@_hdottedline:n	4771, 4775
\l_@@_except_borders_bool	510, 603, 4443, 4447, 4573, 4577	\@@_hdottedline_i:	4766, 4768
\@@_exec_code_before:	1283, 1361	\@@_hdottedline_i:n	4780, 4784
\@@_expand_clist:NN	4191, 4192, 4244	\@@_hline:nn	4457, 4584, 4603
\l_@@_exterior_arraycolsep_bool	498, 791, 1645, 1657, 2452	\@@_hline_i:nn	2716, 4460, 4463
\l_@@_extra_left_margin_dim	513, 619, 1306, 2520	\@@_hline_i_complete:nn	2716, 4567
\l_@@_extra_right_margin_dim	514, 620, 1487, 2565, 3319	\@@_hline_ii:nnnn	4485, 4496, 4529, 4568
\@@_extract_brackets	5399, 5574	\l_@@_hlines_clist	350, 583, 597, 602, 632, 1124, 1126, 1130, 2718, 4582, 4583
\@@_extract_coords_values:	5052, 5059	\l_@@_hpos_of_block_cap_bool	341, 5311, 5314, 5317, 5440, 5510, 5547
\@@_fatal:n ...	15, 284, 1443, 1820, 1849, 2284, 2288, 2290, 2323, 3750, 6444, 6447, 6450	\l_@@_hpos_of_block_tl	339, 340, 5075, 5077, 5079, 5081, 5083, 5085, 5120, 5121, 5123, 5166, 5172, 5182, 5233, 5256, 5260, 5272, 5277, 5304, 5306, 5308, 5310, 5313, 5316, 5513, 5525, 5536, 5540, 5550, 5562
\@@_fatal:nn	16, 1715, 1990	\g_@@_ht_last_row_dim	909, 1204, 1205, 1276, 1277, 1532
\l_@@_fill_tl	333, 5290, 5397, 5399	\g_@@_ht_row_one_dim ..	935, 936, 1201, 1202
\l_@@_final_i_int	2695, 2902, 2907, 2910, 2935, 2943, 2947, 2956, 2964, 3044, 3099, 3180, 3253, 3259, 3262, 3783, 3811, 3879, 3889, 3891	\g_@@_ht_row_zero_dim	929, 930, 1199, 1200, 1527, 2192, 2231
\l_@@_final_j_int	2696, 2903, 2908, 2915, 2920, 2926, 2936, 2944, 2948, 2957, 2965, 3045, 3100, 3133, 3136, 3144, 3370, 3804, 3814, 3816, 3858, 3887	\@@_hvlines_block:nnn	5381, 5638
		\l_@@_hvlines_block_bool ..	345, 5300, 5377
		\@@_i:	4893, 4895, 4896, 4897, 4898, 4907, 4913, 4915, 4916, 4917, 4918, 4923, 4924, 4925, 4926, 4934, 4937, 4939, 4940, 4941, 4993, 4995, 4998, 4999, 5003, 5004, 5029, 5034, 5036, 5038, 5042, 5043, 5054, 5061, 5063, 5065, 5069, 5070
		\g_@@_iddots_int	2687, 3425, 3426
		\l_@@_in_env_bool	274, 402, 1443, 1444
		\c_@@_in_preamble_bool ..	21, 22, 23, 692, 708
		\l_@@_initial_i_int	2693, 2900, 2975, 2978, 3003, 3011, 3015, 3024,

3032, 3042, 3088, 3173, 3219, 3221, 3235, 3241, 3244, 3782, 3783, 3793, 3861, 3871, 3873	\l_@@_left_delim_dim 1291, 1295, 1300, 2267, 2518
\l_@@_initial_j_int 2694, 2901, 2976, 2983, 2988, 2994, 3004, 3012, 3016, 3025, 3033, 3043, 3089, 3111, 3114, 3122, 3309, 3311, 3316, 3362, 3786, 3796, 3798, 3857, 3858, 3869	\g_@@_left_delim_tl 1299, 1430, 1541, 1565, 1633, 1823, 1825, 4805
\l_@@_initial_open_bool 2697, 2977, 2981, 2984, 2991, 2997, 3001, 3017, 3170, 3209, 3216, 3226, 3292, 3299, 3305, 3359, 3411, 3549, 3596, 3780, 3787, 3799, 3855, 3862, 3874, 3932, 4777, 4814	\l_@@_left_margin_dim 511, 613, 1305, 2519, 4796, 5022
\@@_insert_tabularnotes: 2142, 2145	\l_@@_letter_for_dotted_lines_str 802, 813, 814, 1696
\@@_instruction_of_type:nnn 1055, 3639, 3655, 3671, 3694, 3718	\l_@@_letter_vlism_tl 287, 590, 1699
\c_@@_integers_alist_tl 6254, 6265	\l_@@_light_syntax_bool 495, 567, 1308, 1482
\l_@@_inter_dots_dim 486, 487, 2702, 3554, 3561, 3572, 3580, 3587, 3592, 3604, 3612, 4806, 4808, 4836, 4838	\@@_light_syntax_i 2292, 2295
\g_@@_internal_code_after_tl 301, 1760, 1838, 1859, 1867, 1880, 1886, 1892, 1902, 2734, 2735, 4602, 4770, 5144, 5412, 5840, 6046	\@@_line 2733, 3916
\@@_intersect_our_row:nnnn 4174	\@@_line_i:nn 3923, 3930
\@@_intersect_our_row_p:nnnn 4125	\l_@@_line_width_dim 338, 5302, 5585, 5623, 5634, 5640, 5652, 5679, 5699, 5714, 5716, 5726, 5727, 5729, 5740
\@@_j: 4900, 4902, 4903, 4904, 4905, 4910, 4913, 4915, 4918, 4920, 4921, 4923, 4926, 4928, 4929, 4944, 4947, 4949, 4950, 4951, 5006, 5008, 5011, 5013, 5017, 5018, 5031, 5034, 5035, 5037, 5042, 5043, 5055, 5061, 5062, 5064, 5069, 5070	\@@_line_with_light_syntax:n ... 2306, 2310
\l_@@_l_dim 3527, 3528, 3541, 3542, 3554, 3560, 3571, 3579, 3587, 3592, 3604, 3605, 3612, 3613	\@@_line_with_light_syntax_i:n 2305, 2311, 2312
\l_@@_large_nodes_bool 509, 610, 4885, 4889	\@@_math_toggle_token: 167, 977, 2489, 2506, 2535, 2551, 5882, 5886
\g_@@_last_col_found_bool 364, 1246, 1501, 1569, 2418, 2456, 2531, 2643, 5537	\g_@@_max_cell_width_dim 985, 986, 1454, 2393, 4848, 4874
\l_@@_last_col_int 362, 363, 779, 822, 824, 837, 853, 871, 1267, 1270, 1504, 1649, 2603, 2605, 2644, 2646, 3279, 3314, 3636, 3652, 3690, 3714, 5333, 5338, 5339, 5340, 5343, 5372, 5384, 5394, 5406, 5418, 5433, 5464, 5469, 5477, 5493, 5789, 5795, 5801, 6443, 6461	\c_@@_max_l_dim 3541, 3546
\l_@@_last_col_without_value_bool 361, 821, 2645, 6446	\l_@@_medium_nodes_bool 508, 609, 4883, 5484
\l_@@_last_empty_column_int 4699, 4700, 4713, 4719, 4732	\@@_message_hdotsfor: 6452, 6460, 6467, 6475
\l_@@_last_empty_row_int 4681, 4682, 4695, 4716	\@@_msg_new:nn 17, 6352, 6358, 6385, 6394, 6424, 6457, 6464, 6472, 6480, 6485, 6491, 6496, 6501, 6506, 6511, 6516, 6521, 6526, 6532, 6538, 6543, 6549, 6555, 6560, 6567, 6574, 6581, 6587, 6596, 6598, 6604, 6621, 6628, 6633, 6640, 6647, 6654, 6661, 6666, 6676, 6682, 6689, 6696, 6702, 6708, 6716, 6718, 6724, 7003
\l_@@_last_i_tl 6060, 6066, 6069, 6082, 6116, 6129, 6131, 6179, 6186	\@@_msg_new:nnn ... 18, 6338, 6610, 6729, 6739, 6754, 6775, 6793, 6836, 6888, 6939, 6989
\l_@@_last_j_tl 6061, 6067, 6070, 6093, 6095, 6152, 6162, 6169	\@@_msg_redirect_name:nn 19, 797, 1581, 5346, 5349
\l_@@_last_row_int 358, 359, 573, 897, 943, 1136, 1261, 1265, 1272, 1489, 1493, 1496, 1508, 1530, 2301, 2302, 2495, 2496, 2540, 2541, 2648, 3159, 3198, 3668, 3684, 3708, 4311, 4319, 5332, 5335, 5336, 5355, 5372, 5384, 5394, 5406, 5417, 5431, 5492, 5503, 5797, 6710	\@@_multicolumn:nnn 1229, 1932
\l_@@_last_row_without_value_bool 360, 1263, 1491, 2647	\g_@@_multicolumn_cells_seq 325, 1239, 1285, 1944, 2671, 2675, 2676, 4918, 4926, 5048, 5448, 5466
	\g_@@_multicolumn_sizes_seq 326, 1240, 1286, 1946, 2678, 2679, 5049
	\g_@@_name_env_str 291, 296, 297, 1437, 1438, 2322, 2575, 2576, 2584, 2585, 2614, 2625, 2633, 2757, 5810, 6441
	\l_@@_name_str 507, 662, 915, 918, 1017, 1020, 1107, 1110, 2339, 2340, 2364, 2365, 2382, 2383, 2413, 2414, 2439, 2442, 2470, 2473, 5039, 5042, 5066, 5069
	\g_@@_names_seq 273, 659, 661, 7001
	\l_@@_nb_cols_int 5763, 5774, 5782, 5788, 5794, 5800
	\l_@@_nb_rows_int 5762, 5773, 5791
	\@@_newcolumnntype 1143, 1599, 1600
	\@@_node_for_cell: 994, 1001, 1372, 2516, 2566
	\@@_node_for_multicolumn:nn 5050, 5057
	\@@_node_left:nn 6228, 6229, 6288
	\@@_node_position: 1330, 1332, 1336, 1338, 1401, 1403, 1411, 1417
	\@@_node_position_i: 1414, 1418
	\@@_node_right:nn 6238, 6240, 6312

<code>\g_@@_not_empty_cell_bool</code>	307, 992, 999, 2622	<code>\@@_patch_preamble_iv:nnn</code>
<code>\@@_not_in_exterior:nnnn</code>	4166	1678, 1679, 1680, 1769
<code>\@@_not_in_exterior_p:nnnn</code>	4103	<code>\@@_patch_preamble_ix:n</code> 1909, 1927
<code>\l_@@_notes_above_space_dim</code> 502, 503	<code>\@@_patch_preamble_v:nnnn</code>	1681, 1682, 1790
<code>\l_@@_notes_bottomrule_bool</code>	<code>\@@_patch_preamble_vi:n</code> 1683, 1812
.....	680, 840, 865, 2168	<code>\@@_patch_preamble_vii:nn</code>
<code>\l_@@_notes_code_after_tl</code> 678, 2177	1684, 1685, 1686, 1818
<code>\l_@@_notes_code_before_tl</code> 676, 2149	<code>\@@_patch_preamble_viii_i:nn</code>	1831, 1834, 1836
<code>\@@_notes_label_in_list:n</code>	371, 390, 398, 688	<code>\@@_patch_preamble_viii:nn</code>
<code>\@@_notes_label_in_tabular:n</code>	370, 411, 685	1687, 1688, 1689, 1847
<code>\l_@@_notes_para_bool</code>	.. 674, 838, 863, 2153	<code>\@@_patch_preamble_viii_i:nnn</code>	.. 1851, 1873
<code>\@@_notes_style:n</code>	<code>\@@_patch_preamble_x:n</code>
.....	369, 372, 390, 398, 414, 419, 682	1729, 1788, 1810, 1816, 1906, 1930
<code>\l_@@_nullify_dots_bool</code>	<code>\@@_patch_preamble_xi:n</code> 1697, 1898
.....	504, 608, 3643, 3659, 3675, 3699, 3722	<code>\@@_pgf_rect_node:nnn</code> 458, 1415, 5486
<code>\l_@@_number_of_notes_int</code>	368, 405, 415, 425	<code>\@@_pgf_rect_node:nnnn</code>
<code>\@@_old_CT@arc@</code>	1445, 2759	433, 5033, 5060, 5436, 5480, 6215
<code>\@@_old_cdots</code>	1212, 3660	<code>\c_@@_pgfortikzpicture_tl</code>
<code>\@@_old_ddots</code>	1214, 3700	42, 46, 2787, 3949, 4779
<code>\@@_old_dotfill</code>	5826, 5829, 5837	<code>\@@_pgfpointanchor:n</code> 6242, 6247
<code>\@@_old_dotfill:</code>	1233	<code>\@@_pgfpointanchor_i:nn</code> 6250, 6252
<code>\l_@@_old_iRow_int</code>	302, 1257, 2805	<code>\@@_pgfpointanchor_ii:w</code> 6253, 6261
<code>\@@_old_ialign:</code>	1089, 1208, 5327	<code>\@@_pgfpointanchor_iii:w</code> 6274, 6276
<code>\@@_old_iddots</code>	1215, 3723	<code>\@@_picture_position:</code>
<code>\l_@@_old_jCol_int</code>	303, 1259, 2806	1325, 1332, 1338, 1403, 1417, 1418
<code>\@@_old_ldots</code>	1211, 3644	<code>\g_@@_pos_of_blocks_seq</code>
<code>\@@_old_multicolumn</code>	3738	320, 1284, 1456, 1947, 2662, 2666, 2667,
<code>\@@_old_pgfpointanchor</code>	175, 6246, 6250	2710, 2765, 4101, 4337, 4470, 4746, 5362, 5850
<code>\@@_old_pgful@check@rerun</code>	93, 97	<code>\g_@@_pos_of_stroken_blocks_seq</code>
<code>\@@_old_vdots</code>	1213, 3676	322, 1457, 4341, 4474, 5374
<code>\@@_open_x_final_dim:</code>	<code>\g_@@_pos_of_xdots_seq</code>
.....	3127, 3179, 3213, 3371, 3420	321, 1458, 2711, 3040, 4339, 4472
<code>\@@_open_x_initial_dim:</code>	<code>\@@_pre_array:</code> 1255, 1316, 1479
.....	3105, 3172, 3210, 3364, 3414	<code>\@@_pre_array_i:w</code> 1312, 1479
<code>\@@_open_y_final_dim:</code>	3251, 3303, 3369, 3419	<code>\@@_pre_array_ii:</code> 1166, 1287
<code>\@@_open_y_initial_dim:</code>	<code>\@@_pre_code_before:</code> 1318, 1393
.....	3233, 3300, 3361, 3413	<code>\c_@@_preamble_first_col_tl</code> 1638, 2481
<code>\l_@@_parallelize_diags_bool</code>	<code>\c_@@_preamble_last_col_tl</code> 1650, 2526
.....	499, 500, 605, 2684, 3375, 3423	<code>\g_@@_preamble_tl</code>
<code>\@@_patch_m_preamble:n</code>	1432, 1601, 1605, 1609, 1615,
.....	1940, 1968, 2005, 2010, 2076	1629, 1638, 1647, 1650, 1659, 1663, 1703,
<code>\@@_patch_m_preamble_i:n</code>	1712, 1722, 1733, 1746, 1771, 1792, 1814,
.....	1972, 1973, 1974, 1992	1830, 1858, 1866, 1879, 1885, 1900, 1913,
<code>\@@_patch_m_preamble_ii:nn</code>	1920, 1929, 1939, 1941, 1994, 2004, 2009,
.....	1975, 1976, 1977, 2002	2014, 2034, 2055, 2065, 2072, 2081, 2276, 2303
<code>\@@_patch_m_preamble_iii:n</code> 1978, 2007	<code>\@@_pred:n</code>
<code>\@@_patch_m_preamble_iv:nnn</code>	136, 166, 2605, 4378, 4391, 4510, 4523
.....	1979, 1980, 1981, 2012	<code>\@@_provide_pgfsyspdfmark:</code>	... 60, 69, 1427
<code>\@@_patch_m_preamble_ix:n</code> 2061, 2079	<code>\@@_put_box_in_flow:</code> 1567, 2084, 2269
<code>\@@_patch_m_preamble_v:nnnn</code>	1982, 1983, 2032	<code>\@@_put_box_in_flow_bis:nn</code> 1564, 2236
<code>\@@_patch_m_preamble_vi:n</code> 1984, 2053	<code>\@@_put_box_in_flow_i:</code> 2090, 2092
<code>\@@_patch_m_preamble_x:n</code>	<code>\@@_qpoint:n</code>
.....	2000, 2030, 2051, 2056, 2058, 2082	262, 2095, 2097, 2109,
<code>\@@_patch_node_for_cell:</code> 1027, 1372	2125, 2184, 2186, 2202, 2213, 2224, 2873,
<code>\@@_patch_node_for_cell:n</code>	1025, 1051, 1054	2875, 2877, 2879, 2887, 2889, 3122, 3144,
<code>\@@_patch_preamble:n</code>	3173, 3180, 3219, 3221, 3235, 3253, 3309,
.....	1621, 1667, 1705, 1713, 1734, 1765,	3311, 3362, 3370, 3957, 3960, 4211, 4215,
.....	1827, 1843, 1845, 1861, 1869, 1895, 1904, 1924	4231, 4233, 4401, 4403, 4405, 4533, 4535,
<code>\@@_patch_preamble_i:n</code>	1671, 1672, 1673, 1720	4537, 4787, 4791, 4798, 4832, 4835, 4837,
<code>\@@_patch_preamble_ii:nn</code>	4939, 4949, 5427, 5429, 5431, 5433, 5457,
.....	1674, 1675, 1676, 1731	5477, 5506, 5607, 5609, 5616, 5618, 5653,
<code>\@@_patch_preamble_iii:n</code>	1677, 1736, 1744	5655, 5659, 5664, 5666, 5670, 5713, 5715,
<code>\@@_patch_preamble_iii_i:n</code> 1739, 1741	5717, 5724, 5728, 5730, 5863, 5865, 5868,
		5870, 5923, 5925, 6113, 6116, 6154, 6171, 6188

<code>\l_@@_radius_dim</code>	490, 491, 1901, 2701, 3184, 3185, 3621, 4765, 4789, 4833, 4834
<code>\l_@@_real_left_delim_dim</code>	2238, 2253, 2268
<code>\l_@@_real_right_delim_dim</code>	2239, 2265, 2271
<code>\@@_recreate_cell_nodes:</code>	1341, 1397
<code>\g_@@_recreate_cell_nodes_bool</code>	506, 1177, 1341, 1364, 1371, 1376
<code>\@@_rectanglecolor</code> . .	1352, 4026, 4059, 4079
<code>\@@_rectanglecolor:nnn</code> . . .	4032, 4041, 4044
<code>\@@_renew_NC@rewrite@S:</code>	186, 190, 210, 1245
<code>\@@_renew_dots:</code>	1150, 1238
<code>\l_@@_renew_dots_bool</code>	606, 787, 1238, 6369, 6376
<code>\@@_renew_matrix:</code>	782, 786, 5742, 6371, 6375
<code>\l_@@_respect_blocks_bool</code>	4085, 4099, 4122
<code>\@@_restore_iRow_jCol:</code>	2758, 2803
<code>\@@_revtex_array:</code>	1066, 1077
<code>\c_@@_revtex_bool</code>	50, 52, 55, 57, 1076
<code>\l_@@_right_delim_dim</code>	1292, 1296, 1302, 2270, 2563
<code>\g_@@_right_delim_tl</code> . .	1301, 1431, 1559, 1565, 1634, 1826, 1855, 1856, 1877, 1882, 4807
<code>\l_@@_right_margin_dim</code>	512, 615, 1486, 2564, 3318, 4803, 5025
<code>\@@_rotate:</code>	1231, 3909
<code>\g_@@_rotate_bool</code>	279, 952, 979, 1804, 2046, 2508, 2553, 3909, 5166, 5190, 5195, 5255, 5271, 5423
<code>\@@_rotate_cell_box:</code>	940, 979, 1804, 2046, 2508, 2553, 5423
<code>\l_@@_rounded_corners_dim</code>	336, 5294, 5407, 5599, 5600, 5635, 5681, 5738
<code>\@@_roundedrectanglecolor</code>	1353, 4035
<code>\l_@@_row_max_int</code>	328, 2910, 3052, 3069
<code>\l_@@_row_min_int</code>	327, 2978, 3050, 3067
<code>\g_@@_row_of_col_done_bool</code>	306, 1121, 1436, 2348
<code>\g_@@_row_total_int</code> . . .	266, 1242, 1322, 1328, 1399, 1507, 2120, 2219, 2648, 2655, 2823, 3108, 3130, 3824, 4893, 4907, 4934, 5029, 5355, 5443, 5461, 5930, 6069, 6083, 6571
<code>\@@_rowcolor</code>	1354, 3996
<code>\@@_rowcolor:n</code>	4002, 4005
<code>\@@_rowcolor_tabular</code>	1163, 4279
<code>\@@_rowcolors</code>	1355, 4092
<code>\@@_rowcolors:i:nnnn</code>	4126, 4161
<code>\l_@@_rowcolors_restart_bool</code> . . .	4088, 4114
<code>\g_@@_rows_seq</code> .	2298, 2300, 2302, 2304, 2306
<code>\l_@@_rows_tl</code>	4007, 4022, 4052, 4128, 4192, 4217
<code>\l_@@_rules_color_tl</code>	304, 542, 1476, 1477, 6144, 6145
<code>\@@_set_CT@arc@:</code>	169, 1477, 6145
<code>\@@_set_CT@arc@_i:</code>	170, 171
<code>\@@_set_CT@arc@_ii:</code>	170, 173
<code>\@@_set_final_coords:</code>	3078, 3103
<code>\@@_set_final_coords_from_anchor:n</code> . .	3094, 3183, 3214, 3295, 3304, 3374, 3422
<code>\@@_set_initial_coords:</code>	3073, 3092
<code>\@@_set_initial_coords_from_anchor:n</code> .	3083, 3176, 3211, 3294, 3301, 3366, 3416
<code>\@@_set_size:n</code>	5760, 5775
<code>\c_@@_siunitx_loaded_bool</code>	176, 180, 185, 231
<code>\c_@@_size_seq</code>	1265, 1270, 1320, 1321, 1322, 1323, 2651
<code>\l_@@_small_bool</code>	780, 827, 833, 855, 886, 1172, 1820, 1849, 2490, 2536, 2699
<code>\@@_standard_cline</code>	132, 1217
<code>\@@_standard_cline:w</code>	132, 133
<code>\l_@@_standard_cline_bool</code> . .	483, 554, 1216
<code>\c_@@_standard_tl</code>	493, 494, 3448, 4809, 4839
<code>\g_@@_static_num_of_col_int</code>	332, 1576, 1622, 5343, 6476, 6488, 6669
<code>\l_@@_stop_loop_bool</code>	2904, 2905, 2937, 2950, 2959, 2972, 2973, 3005, 3018, 3027
<code>\@@_store_in_tmpb_tl</code>	5577, 5579
<code>\@@_stroke_block:nnn</code>	5369, 5581
<code>\@@_stroke_borders_block:nnn</code> . . .	5391, 5677
<code>\@@_stroke_horizontal:n</code> . .	5705, 5707, 5722
<code>\@@_stroke_vertical:n</code>	5701, 5703, 5711
<code>\@@_sub_matrix:nnnnn</code>	6053, 6055
<code>\@@_sub_matrix_i:nn</code>	6105, 6111
<code>\l_@@_submatrix_extra_height_dim</code>	347, 5976, 6139
<code>\l_@@_submatrix_hlines_clist</code>	352, 5988, 6006, 6178, 6180
<code>\l_@@_submatrix_left_xshift_dim</code>	348, 5978, 6191, 6224
<code>\l_@@_submatrix_name_str</code>	6021, 6073, 6213, 6215, 6227, 6229, 6237, 6240
<code>\g_@@_submatrix_names_seq</code>	324, 2737, 6018, 6022, 6619
<code>\l_@@_submatrix_right_xshift_dim</code>	349, 5980, 6200, 6234
<code>\g_@@_submatrix_seq</code> . . .	331, 1282, 3054, 6044
<code>\l_@@_submatrix_slim_bool</code>	5986, 6081, 6592
<code>\l_@@_submatrix_vlines_clist</code>	353, 5990, 6008, 6161, 6163
<code>\@@_succ:n</code>	161, 165, 1099, 1105, 1131, 1761, 1839, 1918, 1945, 2070, 2097, 2431, 2437, 2442, 2443, 2464, 2468, 2473, 2474, 3144, 3221, 3311, 3370, 4170, 4215, 4231, 4374, 4405, 4448, 4506, 4537, 4578, 4603, 4751, 4753, 4755, 4757, 4798, 4837, 4999, 5003, 5013, 5017, 5431, 5433, 5477, 5616, 5618, 5868, 5870, 5925
<code>\l_@@_suffix_tl</code>	4961, 4972, 4982, 4985, 5034, 5042, 5043, 5061, 5069, 5070
<code>\c_@@_table_collect_begin_tl</code> .	217, 242, 244
<code>\c_@@_table_print_tl</code>	219, 245, 246
<code>\l_@@_tabular_width_dim</code>	277, 1082, 1084, 1661, 2634
<code>\l_@@_tabularnote_tl</code>	367, 842, 867, 2141, 2150
<code>\g_@@_tabularnotes_seq</code>	366, 406, 2156, 2162, 2178
<code>\@@_test_hline_in_block:nnnn</code>	4471, 4473, 4606
<code>\@@_test_hline_in_stroken_block:nnnn</code> .	4475, 4628
<code>\@@_test_if_cell_in_a_block:nn</code>	4685, 4703, 4721, 4741
<code>\@@_test_if_cell_in_block:nnnnnnn</code> . . .	4747, 4749
<code>\@@_test_if_math_mode:</code>	281, 1442, 2586
<code>\@@_test_in_corner_h:</code>	4476, 4504
<code>\@@_test_in_corner_v:</code>	4344, 4372

bool commands:

- \bool_do_until:Nn 2905, 2973
- \bool_gset_false:N 952,
998, 999, 1120, 1246, 1364, 1436, 1460,
1606, 2522, 2569, 4615, 4626, 4637, 4648, 5195
- \bool_gset_true:N
1463, 1764, 2348, 2486, 2531, 2622, 3645,
3661, 3677, 3701, 3724, 3735, 3909, 4336, 4469
- \bool_if:NTF
.. 168, 696, 701, 712, 717, 883, 886, 979,
1094, 1121, 1168, 1172, 1177, 1237, 1238,
1264, 1269, 1283, 1289, 1341, 1343, 1366,
1369, 1371, 1391, 1428, 1443, 1453, 1491,
1569, 1574, 1592, 1597, 1624, 1636, 1804,
1820, 1849, 2046, 2168, 2334, 2351, 2369,
2387, 2400, 2426, 2456, 2461, 2490, 2508,
2536, 2553, 2643, 2645, 2647, 2684, 2699,
2721, 2738, 2741, 2756, 3226, 3228, 3375,
3423, 3643, 3659, 3675, 3699, 3722, 4099,
4122, 4694, 4712, 4730, 4857, 4867, 4889,
5166, 5190, 5255, 5271, 5377, 5423, 5440,
5484, 5510, 5547, 5830, 6403, 6413, 6446, 6592
- \bool_if:nTF ... 185, 375, 402, 1057, 1501,
3059, 3934, 4176, 4443, 4447, 4573, 4577,
4883, 5537, 5927, 5937, 5939, 5952, 5958, 5967
- \bool_lazy_all:nTF
1640, 1652, 2447, 2708, 4608, 4619, 4630, 4641
- \bool_lazy_and:nnTF
..... 230, 1708, 2390, 2491, 3215,
3482, 3741, 4187, 4407, 4539, 5603, 6165, 6182
- \bool_lazy_or:nnTF
.. 524, 991, 1049, 1632, 2118, 2139, 2217,
2539, 3292, 3540, 4168, 4200, 4204, 4254,
4258, 4686, 4704, 4722, 5107, 5112, 5132, 6068
- \bool_lazy_or_p:nn 2494
- \bool_not_p:n
... 1643, 1645, 1655, 1657, 2392, 2450, 2452
- \bool_set:Nn 3296, 4116
- \c_false_bool
1860, 1868, 1881, 1887, 1893, 3639, 3655, 3671
- \g_tmpa_bool
.... 4336, 4345, 4379, 4387, 4392, 4469,
4477, 4511, 4519, 4524, 4615, 4626, 4637, 4648
- \g_tmpb_bool 1606, 1636, 1764
- \l_tmpb_bool ... 4691, 4705, 4723, 4745, 4758
- \c_true_bool 1839

box commands:

- \box_clear_new:N 1170, 1288
- \box_dp:N 908,
928, 965, 984, 1039, 1198, 1207, 1279,
1782, 2025, 2087, 2247, 2260, 3255, 5225, 6118
- \box_gclear_new:N 5154
- \box_grotate:Nn 5192
- \box_ht:N 909, 930, 936, 948,
971, 982, 1031, 1200, 1202, 1205, 1277,
1778, 2021, 2086, 2247, 2260, 3237, 5216, 6115
- \box_move_down:nn 1039
- \box_move_up:nn
..... 76, 78, 80, 1031, 2130, 2195, 2234
- \box_rotate:Nn 942
- \box_set_dp:Nn 964, 983, 2087
- \box_set_ht:Nn 970, 981, 2086
- \box_set_wd:Nn 958

- \box_use:N 426, 949, 1038
- \box_use_drop:N 989, 995, 1014, 1806, 2048,
2089, 2130, 2131, 2136, 2517, 5235, 5532, 5569
- \box_wd:N 427,
959, 986, 993, 1052, 1300, 1302, 2135,
2254, 2266, 2512, 2515, 2557, 2561, 5204, 5837
- \l_tmpa_box
.. 409, 426, 427, 1299, 1300, 1301, 1302,
1536, 2086, 2087, 2089, 2130, 2131, 2247, 2260
- \l_tmpb_box 2240, 2254, 2255, 2266

C

- \c 59, 1628
- \Cdots 1220, 3650, 3653
- \cdots 1153, 1212
- \cellcolor 1162, 1351, 4274
- \chessboardcolors 1358
- \cline 160, 1217, 1218

clist commands:

- \clist_clear:N 4247
- \clist_if_empty:NTF 4343, 4476, 5387
- \clist_if_empty_p:N 2451
- \clist_if_in:NnTF ... 1129, 1613, 1918,
2070, 4453, 4583, 5700, 5702, 5704, 5706, 5725
- \clist_if_in:nnTF 5686
- \clist_map_inline:Nn
... 4194, 4217, 4248, 4653, 5684, 6163, 6180
- \clist_map_inline:nn 2610, 4058, 4107
- \clist_new:N 335, 350, 351, 352, 353, 501
- \clist_put_right:Nn 4265
- \clist_set:Nn 596, 597, 601, 602, 630, 631, 632
- \clist_set_eq:NN 4246
- \l_tmpa_clist 4246, 4248
- \CodeAfter
878, 1234, 2292, 2295, 2485, 2530, 2736, 6744
- \CodeBefore 1425, 6427
- \color 111, 118, 172, 174, 1540,
1558, 3157, 3160, 3163, 3196, 3199, 3202,
3277, 3280, 3284, 3352, 3404, 3822, 3825,
3828, 3897, 3900, 3903, 3920, 3984, 4135,
4136, 4147, 4148, 5161, 5296, 5957, 6297, 6321
- \colorlet 289, 290, 893, 900, 2500, 2544
- \columncolor 1164, 1357, 2747, 4297
- \cr 144, 162, 2479
- \crrcr 2328

cs commands:

- \cs_generate_variant:Nn
..... 58, 164, 3479, 3977, 4879, 4880
- \cs_gset:Npn 111, 118, 4872
- \cs_gset_eq:NN ... 69, 247, 1191, 1445, 2759
- \cs_if_exist:NTF 57, 188,
1257, 1259, 1406, 1446, 1449, 2805, 2806,
2827, 2940, 2953, 3008, 3021, 3110, 3132,
3240, 3258, 3790, 3808, 3865, 3883, 4859,
4912, 5445, 5463, 5932, 6085, 6092, 6121, 6128
- \cs_if_exist_p:N . 232, 525, 4688, 4707, 4725
- \cs_if_free:NTF
..... 65, 3152, 3191, 3272, 3347, 3399
- \cs_if_free_p:N 3936, 3938
- \cs_new_protected:Npx 2785, 3947, 4775
- \cs_set:Nn 682, 685, 688
- \cs_set:Npn 107, 108, 114,
115, 120, 132, 133, 147, 149, 150, 172, 174,

\Hspace 1226
\hspace 3736
\hss 1682, 1983

I

\ialign 1089, 1185, 1208, 5327
\iddots 1223, 3707, 3708, 3713, 3714
\iddots 71, 1156, 1215

if commands:

\if_mode_math: 283
\IfBooleanTF 1479
\ifnum 122, 4587, 4604
\ifstandalone 1449
\ignorespaces 1966

int commands:

\int_case:nnTF 3681, 3687, 3705, 3711
\int_compare:nNnTF 135, 136, 152,
880, 881, 890, 897, 933, 943, 1134, 1136,
1261, 1267, 1272, 1489, 1493, 1504, 1508,
1509, 1576, 1942, 1964, 2151, 2190, 2229,
2301, 2329, 2537, 2915, 2922, 2926, 2928,
2983, 2990, 2994, 2996, 3159, 3198, 3279,
3314, 3316, 3824, 3899, 4163, 4208, 4226,
4262, 4293, 4310, 4311, 4318, 4319, 4323,
4751, 4753, 4755, 4757, 5160, 5197, 5209,
5503, 5535, 5539, 5612, 5614, 5785, 5787,
5789, 5793, 5795, 5797, 5799, 5801, 6149, 6151
\int_compare_p:n
3061, 3062, 3063, 3064, 4178, 4180, 5604, 5605
\int_do_until:nNnn 4119
\int_gadd:Nn 1963
\int_gincr:N .. 879, 906, 1452, 1728, 1787,
1809, 1815, 2424, 2532, 3377, 3425, 4854, 5141
\int_if_even:nTF 4067, 6268
\int_if_odd_p:n 4116
\int_incr:N 405, 1738
\int_min:nn
2873, 2875, 2877, 2879, 2887, 2889, 3069, 3070
\int_step_inline:nn
..... 2871, 4063, 4065, 6162, 6179
\int_step_inline:nnnn 4683, 4701, 4716, 4719
\c_zero_int 1821, 3970, 4169

iow commands:

\iow_newline:
... 1589, 2660, 2668, 2677, 2680, 2752, 4674
\iow_now:Nn 62, 88, 1585, 1586, 1591
\iow_shipout:Nn 4869, 4870, 4876
\item 2156, 2162

K

\kern 81

keys commands:

\keys_define:nn . 520, 540, 547, 625, 672,
724, 731, 775, 817, 831, 848, 861, 1374,
3727, 4074, 4083, 4843, 5073, 5288, 5629,
5735, 5765, 5889, 5894, 5974, 5995, 6004, 6364
\l_keys_key_str
..... 6340, 6529, 6535, 6623, 6726,
6731, 6741, 6756, 6777, 6795, 6838, 6890, 6941
\keys_set:nn 199,
550, 552, 566, 806, 809, 816, 1378, 1386,
1473, 1474, 2616, 2626, 2635, 2762, 3162,
3201, 3282, 3351, 3403, 3827, 3902, 3919,
4078, 4097, 4856, 5364, 5896, 5900, 6027, 6074

\keys_set_known:nn
..... 3693, 3717, 5124, 5586, 5641, 5680
\keys_set_known:nnN 5778
\l_keys_value_tl 6576, 6652, 6659, 6991

L

\Ldots 1219, 3634, 3637
\ldots 1152, 1211
\leaders 140, 157
\left 1541, 2243, 2258, 5958, 6298, 6322
legacy commands:
\legacy_if:nTF 656
\line 2733, 6606, 6752

M

\makebox 1806, 2048
\mathinner 73
mode commands:
\mode_leave_vertical: 1441, 1776, 2019
msg commands:
\msg_error:nn 12
\msg_error:nnn 13
\msg_error:nnnn 14, 5345, 5352, 5356
\msg_fatal:nn 15
\msg_fatal:nnn 16
\msg_new:nnn 17
\msg_new:nnnn 18
\msg_redirect_name:nnn 20
\multicolumn 1229, 3738, 3747, 3753, 3775
\multispan 136, 137, 153, 154, 1934
\myfiledate 6
\myfileversion 7

N

\newcolumnntype 254, 255, 256
\newcounter 365
\NewDocumentCommand
.. 377, 400, 815, 1384, 2761, 3631, 3647,
3663, 3679, 3703, 3761, 3838, 3916, 3996,
4011, 4026, 4035, 4056, 4061, 4076, 4092,
4268, 4279, 4291, 5574, 5771, 5819, 6034, 6050
\NewDocumentEnvironment 1424,
2273, 2282, 2572, 2582, 2612, 2623, 2631, 4852
\NewExpandableDocumentCommand 260, 5094
\newlist 381, 392
\NiceArray 2628, 2637
\NiceArrayWithDelims 2577, 2587
nicematrix commands:
\g_nicematrix_code_after_tl .. 299, 665,
2297, 2739, 2742, 5367, 5379, 5389, 5907, 5914
\g_nicematrix_code_before_tl
... 1253, 2744, 2751, 4272, 4283, 4295, 5400
\NiceMatrixLastEnv 260
\NiceMatrixOptions 815, 6796, 6996
\NiceMatrixoptions 6656
\noalign 110, 117, 122, 145, 1116, 1191, 4587, 4765
\nobreak 395
\normalbaselines 1171
\NotEmpty 1236
\nulldelimiterspace 2254, 2266, 5943, 6141
\nullfont 5954, 5961, 6294, 6301, 6318, 6325
\numexpr 165, 166

O	
\omit	135, 2331, 2347, 2423, 5904
\OnlyMainNiceMatrix	1232, 4302
P	
\par	2150, 2158
peek commands:	
\peek_meaning:NTF	170, 407, 1146
\peek_meaning_ignore_spaces:NTF	2275, 4590
\peek_meaning_remove_ignore_spaces:NTF	160
\peek_remove_spaces:n	4270, 4281, 5096, 6036, 6052
\pgfdeclareshape	2808
\pgfextracty	5506
\pgfgetlastxy	469
\pgfpathcircle	3619
\pgfpathlineto	4424, 4430, 4554, 4560, 5661, 5672, 5719, 5732, 5872, 6156, 6173, 6207
\pgfpathmoveto	4423, 4429, 4553, 4559, 5660, 5671, 5718, 5731, 5867, 6155, 6172, 6198
\pgfpathrectanglecorners	4236, 4417, 4547, 5620
\pgfpointadd	467
\pgfpointanchor	175, 263, 2831, 2844, 3085, 3096, 3113, 3135, 3243, 3261, 4915, 4923, 5450, 5468, 5488, 5490, 5507, 5544, 5935, 6088, 6095, 6124, 6131, 6242, 6246
\pgfpointdiff	468, 1332, 1338, 1403, 1417, 1418
\pgfpointlineatime	3489
\pgfpointorigin	2338, 2469, 2816
\pgfpointscale	467
\pgfpointshapeborder	3957, 3960
\pgfrememberpicturepositiononpagetrue	911, 1005, 1104, 2337, 2361, 2379, 2410, 2436, 2467, 2794, 2821, 2870, 3446, 3956, 4399, 4531, 4786, 4831, 4958, 4968, 4979, 5425, 5588, 5649, 5696, 5862, 5921, 6076
\pgfscope	2829, 2842, 3486, 5879
\pgfset	436, 461, 1006, 5521, 5558, 5878, 5942, 6078
\pgfsetbaseline	1004
\pgfsetcornersarced	4235, 5596
\pgfsetlinewidth	4433, 4563, 5623, 5652, 5699, 6143
\pgfsetrectcap	4434, 4564
\pgfsetroundcap	5875
\pgfsetstrokecolor	5594
\pgfsyspdfmark	65, 66
\pgftransformrotate	3493
\pgftransformshift	442, 467, 2830, 2843, 2881, 3487, 5520, 5542, 5880, 5884, 5944, 6221, 6231
\pgfusepath	3513, 3523, 3987, 4139, 4151, 4420, 5624
\pgfusepathqfill	3625, 4550
\pgfusepathqstroke	4435, 4565, 5662, 5673, 5720, 5733, 5876, 6157, 6174, 6208
\phantom	3644, 3660, 3676, 3700, 3723
\pNiceMatrix	5745
prg commands:	
\prg_do_nothing:	69, 186, 229, 238, 247, 537, 1191, 2736
\prg_new_conditional:Nnn	4166, 4174
\prg_replicate:nn	415, 2419, 2420, 3775, 4425, 4555, 5788, 5791, 5794, 5800
\prg_return_false:	4171, 4183
\prg_return_true:	4172, 4182
\ProcessKeysOptions	6384
\ProvideDocumentCommand	71
\ProvidesExplPackage	4
Q	
\quad	396
quark commands:	
\q_stop	132, 133, 149, 150, 171, 173, 1367, 1389, 1477, 1621, 1693, 1763, 1853, 1875, 1940, 1988, 2292, 2295, 3910, 3924, 3925, 3991, 4046, 4051, 4111, 4198, 4199, 4221, 4222, 4252, 4253, 5052, 5059, 5099, 5100, 5104, 5399, 5579, 5602, 5611, 5642, 5645, 5689, 5692, 5760, 5775, 6038, 6043, 6062, 6065, 6145, 6253, 6261
R	
\rectanglecolor	1352, 2746, 4285
\refstepcounter	424
regex commands:	
\regex_const:Nn	59
\regex_match:nnTF	6016
\regex_replace_all:NnN	1626
\relax	165, 166
\renewcommand	192, 212
\RenewDocumentEnvironment	5744, 5747, 5750, 5753, 5756
\RequirePackage	1, 3, 9, 10, 11
\right	1559, 2250, 2262, 5967, 6306, 6330
\rotate	1231
\roundedrectanglecolor	1353, 5402
\rowcolor	1163, 1354
\rowcolors	1355
S	
\savedanchor	2810
\savenotes	1428
scan commands:	
\scan_stop:	2740
\scriptstyle	886, 2490, 2536, 3474, 3475, 3509, 3519
seq commands:	
\seq_clear:N	1619
\seq_clear_new:N	4652
\seq_count:N	2302, 3973
\seq_gclear:N	1239, 1240, 1282, 1284, 1455, 1456, 1457, 1458, 2178, 2737
\seq_gclear_new:N	1285, 1286, 1363, 2298, 2314
\seq_gpop_left:NN	2304, 2316
\seq_gput_left:Nn	661, 1944, 1946, 5362
\seq_gput_right:Nn	406, 1701, 1947, 3040, 3972, 5230, 5242, 5374, 5850, 6022, 6044
\seq_gset_from_clist:Nn	2651, 2666, 2675, 2678
\seq_gset_map_x:NNn	2765
\seq_gset_split:Nnn	2300, 2315
\seq_if_empty:NTF	2138, 2662, 2671, 4668
\seq_if_empty_p:N	2710, 2711, 2712, 4188
\seq_if_in:NnTF	659, 4228, 4376, 4382, 4389, 4508, 4514, 4521, 4918, 4926, 5448, 5466, 6018, 6441
\seq_item:Nn	1265, 1270, 1320, 1321, 1322, 1323
\seq_map_function:NN	2306
\seq_map_indexed_inline:Nn	3968, 3982

<code>\seq_map_inline:Nn</code>	2156, 2162, 2318, 3054, 4126, 4337, 4339, 4341, 4470, 4472, 4474, 4746, 5328, 6147
<code>\seq_mapthread_function:NNN</code>	5047
<code>\seq_new:N</code>	273, 288, 319, 320, 321, 322, 323, 324, 325, 326, 331, 366, 6431
<code>\seq_put_right:Nn</code>	4733
<code>\seq_set_eq:NN</code>	4101
<code>\seq_set_filter:NNn</code>	4102, 4124
<code>\seq_set_from_clist:Nn</code>	4672, 6432
<code>\seq_set_map_x:NNn</code>	6437
<code>\seq_use:Nnnn</code>	2667, 2676, 2679, 4673, 6619, 7001
<code>\l_tmpa_seq</code>	4102, 4124
<code>\l_tmpb_seq</code>	4101, 4102, 4124, 4126
<code>\setlist</code>	382, 393, 697, 702, 713, 718
siunitx commands:	
<code>\siunitx_cell_begin:w</code>	188, 200, 232
<code>\siunitx_cell_end:</code>	203
skip commands:	
<code>\skip_gadd:Nn</code>	2395
<code>\skip_gset:Nn</code>	2386
<code>\skip_gset_eq:NN</code>	2393, 2394
<code>\skip_horizontal:N</code>	141, 158, 1305, 1306, 1486, 1487, 1511, 1512, 1548, 1549, 1552, 1553, 1571, 1572, 1610, 1616, 1704, 1901, 1914, 1921, 2066, 2073, 2267, 2268, 2270, 2271, 2342, 2343, 2355, 2357, 2373, 2375, 2397, 2404, 2406, 2425, 2430, 2432, 2460, 2518, 2519, 2520, 2523, 2558, 2563, 2564, 2565
<code>\skip_horizontal:n</code>	427, 1052, 1750, 2455
<code>\skip_vertical:N</code>	145, 1098, 1100, 1544, 1555, 2147, 2172, 4765
<code>\skip_vertical:n</code>	948, 4597
<code>\g_tmpa_skip</code>	2386, 2393, 2394, 2395, 2397, 2425
<code>\c_zero_skip</code>	1196
<code>\space</code>	296, 297
<code>\stepcounter</code>	413, 418
str commands:	
<code>\c_backslash_str</code>	296
<code>\c_colon_str</code>	814
<code>\str_case:nn</code>	5513, 5525, 5550, 5562, 6192, 6201
<code>\str_case:nnTF</code>	1516, 1669, 1970, 2112, 4655, 6265, 6278
<code>\str_clear_new:N</code>	6073
<code>\str_gclear:N</code>	2757
<code>\str_gset:Nn</code>	1438, 2576, 2585, 2614, 2625, 2633, 5810
<code>\str_if_empty:NTF</code>	915, 1017, 1107, 1437, 2339, 2364, 2382, 2413, 2439, 2470, 2575, 2584, 5039, 5066, 6213, 6227, 6237
<code>\str_if_eq:nnTF</code>	96, 295, 1087, 1696, 1699, 1743, 1763, 1823, 1855, 1877, 1908, 2060, 2287, 2289, 2322, 5592, 5912
<code>\str_if_eq_p:nn</code>	526, 1633, 1634, 1709, 4202, 4206, 4256, 4260, 5109, 5114
<code>\str_if_in:NnTF</code>	2100, 2204
<code>\str_new:N</code>	291, 507, 813
<code>\str_range:Nnn</code>	2104, 2208
<code>\str_set:Nn</code>	658, 802, 6021
<code>\str_set_eq:NN</code>	662, 814
<code>\l_tmpa_str</code>	658, 659, 661, 662
<code>\strut</code>	2156, 2162
<code>\strutbox</code>	3237, 3255, 6115, 6118
<code>\SubMatrix</code>	1359, 2720, 6047, 6545, 6577, 6584, 6590, 6594, 6612, 6759
sys commands:	
<code>\sys_if_engine_xetex_p:</code>	1049
<code>\sys_if_output_dvi_p:</code>	1049
T	
<code>\tabcolsep</code>	1080, 1548, 1552
<code>\tabskip</code>	1196
<code>\tabularnote</code>	377, 400, 407, 6684, 6704
<code>\tabularnotes</code>	2161
T _E X and L ^A T _E X 2 _ε commands:	
<code>\@@endpbox</code>	1958
<code>\@@startpbox</code>	1957
<code>\@BTnormal</code>	1169
<code>\@acol</code>	1070
<code>\@acoll</code>	1068
<code>\@acolr</code>	1069
<code>\@array@array</code>	1072
<code>\@arrayacol</code>	1068, 1069, 1070
<code>\@arstrut</code>	1959
<code>\@arstrutbox</code>	908, 909, 948, 1198, 1200, 1202, 1205, 1207, 1778, 1782, 2021, 2025
<code>\@currenvir</code>	5912
<code>\@depth</code>	5964, 6303, 6327
<code>\@endpbox</code>	1958
<code>\@gobblethree</code>	66
<code>\@halignto</code>	1071, 1083, 1084
<code>\@height</code>	125, 140, 157, 5962, 6302, 6326
<code>\@ifclassloaded</code>	51, 54, 6405, 6415
<code>\@ifnextchar</code>	1244
<code>\@ifpackageloaded</code>	30, 33, 36, 39, 87, 103, 179, 6408, 6418
<code>\@mainaux</code>	62, 88, 1585, 1586, 1591, 4869, 4870, 4876
<code>\@mkpream</code>	1941
<code>\@preamble</code>	1960
<code>\@sharp</code>	1955
<code>\@startpbox</code>	1957
<code>\@tabarray</code>	1085
<code>\@tempswafalse</code>	1603, 1938
<code>\@tempswatrue</code>	1602, 1937
<code>\@temptokena</code>	194, 196, 214, 216, 237, 240, 1601, 1621, 1936, 1940
<code>\@whiles</code>	1603, 1938
<code>\@width</code>	125, 5965, 6304, 6328
<code>\@xhline</code>	128
<code>\bBigg@</code>	1299, 1301
<code>\c@MaxMatrixCols</code>	2604, 6469
<code>\c@tabularnote</code>	2140, 2151, 2179
<code>\col@sep</code>	1080, 1081, 1511, 1572, 2342, 2395, 2455, 2523, 2558, 3124, 3146
<code>\CT@arc</code>	107, 108
<code>\CT@arc@</code>	106, 111, 126, 139, 156, 172, 174, 1445, 2173, 2759, 4432, 4562, 4830, 5593, 5651, 5698, 5874, 6146
<code>\CT@dr</code>	114, 115
<code>\CT@drsc@</code>	113, 118, 4409, 4412, 4541, 4544
<code>\CT@everycr</code>	1189
<code>\CT@row@color</code>	1191
<code>\if@temp</code>	1603, 1938
<code>\NC@</code>	1145
<code>\NC@find</code>	205, 221, 238

<code>\NC@list</code>	1603, 1938	<code>\tl_if_eq:NNTF</code>	3448
<code>\NC@rewrite@S</code>	192, 212, 239	<code>\tl_if_eq:NnTF</code> ..	1126, 1607, 1911, 2063,
<code>\new@ifnextchar</code>	1244		2088, 2199, 4452, 4582, 4805, 4807, 6161, 6178
<code>\newcol@</code>	1147, 1148	<code>\tl_if_eq:nnTF</code> ...	651, 793, 1853, 1875, 3969
<code>\nicematrix@redefine@check@rerun</code> ..	88, 91	<code>\tl_if_exist:NNTF</code>	1461
<code>\pgf@relevantforpicturesizefalse</code>		<code>\tl_if_in:NnTF</code>	4110, 4197, 4220, 4251
	1327, 2795, 2822, 3447,	<code>\tl_if_in:nnTF</code>	1840, 1850, 1865
	3616, 3981, 4106, 4400, 4532, 4959, 4969,	<code>\tl_if_single_token:nTF</code>	589, 801
	4980, 5426, 5589, 5650, 5697, 5861, 5922, 6077	<code>\tl_if_single_token_p:n</code>	58
<code>\pgfsys@getposition</code>		<code>\tl_item:Nn</code>	243, 244, 246
	1325, 1330, 1336, 1401, 1409, 1412	<code>\tl_map_inline:nn</code>	2285
<code>\pgfsys@markposition</code>		<code>\tl_new:N</code>	242, 245,
	1034, 1042, 1099, 1181, 1324,		268, 280, 287, 299, 300, 301, 304, 308, 333,
	2335, 2356, 2374, 2405, 2431, 2463, 2837, 2850		334, 337, 339, 342, 367, 492, 496, 515, 517, 518
<code>\pgfutil@check@rerun</code>	93, 94	<code>\tl_put_left:Nn</code>	1169, 1372
<code>\reserved@a</code>	127	<code>\tl_put_right:Nn</code>	641, 1179, 1274, 1314, 1470
<code>\rvtx@ifformat@geq</code>	57	<code>\tl_range:nnn</code>	96
<code>\set@color</code>	5160, 5422	<code>\tl_set:Nn</code>	243, 4052, 4053, 4112,
<code>\set@typeset@protect</code>	1956		4128, 4207, 4209, 4225, 4227, 4261, 4263,
<code>\tikz@library@external@loaded</code>	1446		4332, 5125, 5613, 5646, 5647, 5693, 5694
tex commands:		<code>\tl_set_eq:NN</code>	494, 4049, 4050, 4210,
<code>\tex_mkern:D</code>	75, 77, 79, 82		4348, 4480, 4809, 4839, 5121, 5643, 5644,
<code>\tex_the:D</code>	196, 216		5690, 5691, 6041, 6042, 6063, 6064, 6066, 6067
<code>\textfont</code>	2126	<code>\tl_set_rescan:Nnn</code>	
<code>\textit</code>	369		2299, 3630, 3760, 3837, 3915
<code>\textsuperscript</code>	370, 371	<code>\tl_to_str:n</code>	6438
<code>\the</code>	165, 166, 1603, 1621, 1938, 1940	<code>\g_tmpa_tl</code>	240, 243, 246
<code>\thetabularnote</code>	372	tmpc commands:	
<code>\tikzexternaldisable</code>	1448	<code>\l_tmpc_dim</code>	314, 2878, 2882,
<code>\tikzset</code>	1345, 1450, 2723		4213, 4214, 4237, 4406, 4414, 4419, 4424,
tl commands:			4430, 4538, 4549, 4554, 4560, 5432, 5438,
<code>\tl_clear:N</code>	4358, 4369, 4490, 4501, 5584		5482, 5610, 5621, 5716, 5719, 5869, 5872, 5880
<code>\tl_clear_new:N</code>	4047, 4048, 4095,	<code>\l_tmpc_int</code>	4117, 4118, 4119
	4333, 4466, 6039, 6040, 6058, 6059, 6060, 6061	<code>\l_tmpc_tl</code>	4047, 4049, 4052,
<code>\tl_const:Nn</code>			4210, 4229, 4333, 4347, 4348, 4351, 4356,
	42, 43, 46, 47, 493, 2481, 2526, 6254		4358, 4362, 4367, 4369, 4466, 4479, 4480,
<code>\tl_count:n</code>	2107, 2211		4483, 4488, 4490, 4494, 4499, 4501, 5643,
<code>\tl_gclear:N</code> 1466, 1605, 1939, 2735, 2742, 3986			5655, 5668, 5690, 5707, 5713, 6039, 6041, 6045
<code>\tl_gclear_new:N</code>		tmpd commands:	
	1247, 1248, 1249, 1250, 1251, 1252, 1253, 1459	<code>\l_tmpd_dim</code>	
<code>\tl_gput_left:Nn</code>	1057, 1638, 4295		315, 2880, 2883, 4234, 4237, 4415,
<code>\tl_gput_right:Nn</code>	1057, 1650,		4419, 4545, 4549, 5434, 5438, 5460, 5471,
	1703, 1746, 1760, 1838, 1859, 1867, 1880,		5475, 5478, 5482, 5619, 5622, 5871, 5872, 5884
	1886, 1892, 1902, 2649, 2664, 2673, 2748,	<code>\l_tmpd_tl</code>	4048, 4050, 4053, 5644,
	3763, 3840, 3975, 4272, 4283, 4602, 4670,		5657, 5666, 5691, 5703, 5724, 6040, 6042, 6045
	4770, 5144, 5367, 5379, 5389, 5400, 5412, 5840	token commands:	
<code>\tl_gset:Nn</code>		<code>\token_to_str:N</code>	6426, 6427,
	240, 244, 246, 1430, 1431, 1432, 1588,		6455, 6540, 6545, 6551, 6576, 6584, 6590,
	1609, 1615, 1825, 1826, 1856, 1882, 2750, 3973		6594, 6606, 6612, 6630, 6643, 6684, 6691,
<code>\tl_if_blank:nTF</code>	3998, 4001, 4013,		6704, 6721, 6731, 6743, 6752, 6758, 6796, 6996
	4016, 4028, 4031, 4037, 4040, 4132, 4144, 5098	U	
<code>\tl_if_blank_p:n</code>		<code>\unskip</code>	2166
	4201, 4205, 4255, 4259, 4409, 4541, 5108, 5113	use commands:	
<code>\tl_if_empty:NNTF</code>	1124, 1467, 1476, 1539,	<code>\use:N</code>	1060,
	1557, 2150, 2718, 2719, 2744, 4223, 4224,		1309, 1310, 1464, 1483, 1484, 2599, 2619, 3985
	4347, 4351, 4362, 4479, 4483, 4494, 5119,	<code>\use:n</code>	
	5159, 5365, 5397, 5590, 5956, 6144, 6296, 6320		3921, 4302, 5169, 5179, 5257, 5274, 5779, 6249
<code>\tl_if_empty:nTF</code>		<code>\usepackage</code>	6410, 6420
	639, 777, 819, 835, 851, 869,	<code>\usepgfmodule</code>	2
	2284, 2311, 3163, 3202, 3283, 3352, 3404,	V	
	3828, 3903, 3920, 4134, 4146, 6013, 6263, 6454	vbox commands:	
<code>\tl_if_empty_p:N</code>	1644, 1656, 3483, 3484	<code>\vbox:n</code>	81
<code>\tl_if_empty_p:n</code>	2141		

<code>\vbox_set_top:Nn</code>	945	<code>\vNiceMatrix</code>	5748
<code>\vbox_to_ht:nn</code>	447, 474, 2246, 2259	<code>\vrule</code>	124, 1778, 1782, 2021, 2025
<code>\vbox_to_zero:n</code>	947	<code>\vskip</code>	123
<code>\vcenter</code>	1542, 2244, 5959, 6299, 6323, 6721	<code>\vtop</code>	1096
<code>\Vdots</code>	1221, 3666, 3669		
<code>\vdots</code>	1154, 1213		X
<code>\Vdotsfor</code>	1228	<code>\xglobal</code>	893, 900, 2500, 2544
<code>\vfill</code>	450, 476		
<code>\vline</code>	126		Z
<code>\VNiceMatrix</code>	5751	<code>\Z</code>	6016

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	7
5.1.1	The vertical rules	7
5.1.2	The command <code>\cline</code>	8
5.2	The thickness and the color of the rules	8
5.3	The tools of nicematrix for the rules	9
5.3.1	The keys <code>hlines</code> and <code>vlines</code>	9
5.3.2	The key <code>hvlines</code>	9
5.3.3	The (empty) corners	10
5.4	The command <code>\diagbox</code>	11
5.5	Dotted rules	11
6	The color of the rows and columns	11
6.1	Use of <code>colortbl</code>	11
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	12
6.3	Color tools with the syntax of <code>colortbl</code>	15
7	The width of the columns	16
8	The exterior rows and columns	17
9	The continuous dotted lines	19
9.1	The option <code>nullify-dots</code>	20
9.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	21
9.3	How to generate the continuous dotted lines transparently	22
9.4	The labels of the dotted lines	22
9.5	Customisation of the dotted lines	22
9.6	The dotted lines and the rules	23

10	The <code>\CodeAfter</code>	24
10.1	The command <code>\line</code> in the <code>\CodeAfter</code>	24
10.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	25
11	The notes in the tabulars	26
11.1	The footnotes	26
11.2	The notes of tabular	27
11.3	Customisation of the tabular notes	28
11.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	30
12	Other features	31
12.1	Use of the column type <code>S</code> of <code>siunitx</code>	31
12.2	Alignment option in <code>{NiceMatrix}</code>	31
12.3	The command <code>\rotate</code>	31
12.4	The option <code>small</code>	32
12.5	The counters <code>iRow</code> and <code>jCol</code>	32
12.6	The option <code>light-syntax</code>	33
12.7	Color of the delimiters	33
12.8	The environment <code>{NiceArrayWithDelims}</code>	33
13	Use of Tikz with <code>nicematrix</code>	34
13.1	The nodes corresponding to the contents of the cells	34
13.2	The “medium nodes” and the “large nodes”	34
13.3	The nodes which indicate the position of the rules	36
13.4	The nodes corresponding to the command <code>\SubMatrix</code>	37
14	API for the developers	37
15	Technical remarks	38
15.1	Definition of new column types	38
15.2	Diagonal lines	38
15.3	The “empty” cells	39
15.4	The option <code>exterior-arraycolsep</code>	39
15.5	Incompatibilities	40
16	Examples	40
16.1	Notes in the tabulars	40
16.2	Dotted lines	41
16.3	Dotted lines which are no longer dotted	42
16.4	Stacks of matrices	43
16.5	How to highlight cells of a matrix	46
16.6	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	48
17	Implementation	49
17.1	The redefinition of <code>\multicolumn</code>	99
18	History	206
	Index	212