The package nicematrix*

F. Pantigny
fpantigny@wanadoo.fr

March 11, 2022

Abstract

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

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

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

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

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

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

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

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

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

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

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

*This document corresponds to the version 6.8 of nicematrix, at the date of 2022/03/11.

[1] The 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`

[2] If 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.

```
{NiceTabular}
{NiceTabular*}
{NiceTabularX}
{NiceArray}
{pNiceArray}
{bNiceArray}
{BNiceArray}
{vNiceArray}
{VNiceArray}
{NiceMatrix}
{pNiceMatrix}
{bNiceMatrix}
{BNiceMatrix}
{vNiceMatrix}
{VNiceMatrix}
```

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

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

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

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

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

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

2 The vertical space between the rows

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

```
$\begin{pmatrix}
\frac12 & -\frac12 \\
\frac13 & \frac14 \\
\end{pmatrix}$
```

$$\begin{pmatrix} \frac12 & -\frac12 \\ \frac13 & \frac14 \end{pmatrix}$$

Inspired by the package cellspace which deals with that problem, the package nicematrix provides two keys `cell-space-top-limit` and `cell-space-bottom-limit` similar to the parameters `\cellspacetoplimit` and `\cellspacebottomlimit` of cellspace.
There is also a key `cell-space-limits` to set both parameters at once.
The initial value of these parameters is 0 pt in order to have for the environments of nicematrix the same behaviour as those of array and amsmath. However, a value of 1 pt would probably be a good choice and we suggest to set them with `\NiceMatrixOptions`.[4]

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

```
$\begin{pNiceMatrix}
\frac12 & -\frac12 \\
\frac13 & \frac14 \\
\end{pNiceMatrix}$
```

$$\begin{pmatrix} \frac12 & -\frac12 \\ \frac13 & \frac14 \end{pmatrix}$$

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

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

3 The vertical position of the arrays

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

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

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

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

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

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

1. an item
2. $\begin{array}{lcccccc} \hline n & 0 & 1 & 2 & 3 & 4 & 5 \\ u_n & 1 & 2 & 4 & 8 & 16 & 32 \\ \hline \end{array}$

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

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

1. an item
2. $\begin{array}{lcccccc} \hline n & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline u_n & 1 & 2 & 4 & 8 & 16 & 32 \\ \hline \end{array}$

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}

$A=\begin{pNiceArray}{cc|cc}[baseline=line-3]
\dfrac1A & \dfrac1B & 0 & 0 \\
\dfrac1C & \dfrac1D & 0 & 0 \\
\hline
0 & 0 & A & B \\
0 & 0 & D & D \\
\end{pNiceArray}$
```

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

[5] The extension booktabs is *not* loaded by nicematrix.

4 The blocks

4.1 General case

In the environments of nicematrix, it's possible to use the command `\Block` in order to place an element in the center of a rectangle of merged cells of the array.[6]
The command `\Block` must be used in the upper leftmost cell of the array with two arguments.

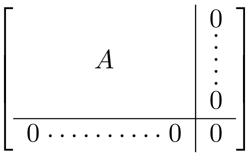
- The first argument is the size of the block with the syntax i-j where i is the number of rows of the block and j its number of columns.

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

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

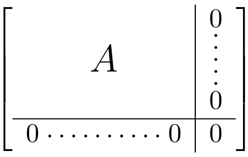
Here is an example of utilisation of the command `\Block` in mathematical matrices.

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



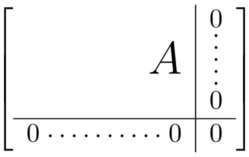
One may wish to raise the size of the "A" placed in the block of the previous example. Since this element is composed in math mode, it's not possible to use directly a command like `\large`, `\Large` and `\LARGE`. That's why the command `\Block` provides an option between angle brackets to specify some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

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



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

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



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:

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

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

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;
- the key `fill` takes in as value a color and fills the block with that color;
- the key `draw` takes in as value a color and strokes the frame of the block with that color (the default value of that key is the current color of the rules of the array);
- the key `color` takes in as value a color and apply that color the content of the block but draws also the frame of the block with that color;
- the key `line-width` is the width (thickness) of the frame (this key should be used only when the key `draw` or the key `hvlines` is in force);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt[8]);
- the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`);
- the keys `hlines`, `vlines` and `hvlines` draw all the corresponding rules in the block;
- when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz[9] by using as options the value of that key `tikz` (which must be a list of keys allowed for a Tikz path). For examples, cf. p. 47;
- the key `name` provides a name to the rectangular Tikz node corresponding to the block; it's possible to use that name with Tikz in the `\CodeAfter` of the environment (cf. p. 28);
- **New 6.5** the key `respect-arraystretch` prevents the setting of `\arraystretch` to 1 at the beginning of the block (which is the behaviour by default) ;
- the key `borders` provides the ability to draw only some borders of the blocks; the value of that key is a (comma-separated) list of elements covered by `left`, `right`, `top` and `bottom`;
  Nouveau 6.7 it's possible, in fact, in the list which is the value of the key `borders`, to add an entry of the form `tikz={`*list*`}` where *list* is a list of couples *key=value* of Tikz specifying the graphical characteristics of the lines that will be drawn (for an example, see p. 50).

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

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

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

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

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

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

4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

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

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

- The specification of the horizontal position provided by the type of column (`c`, `r` or `l`) is taken into account for the blocks.

- The specifications of font specified for the column by a construction `>{...}` in the preamble of the array are taken into account for the mono-column blocks of that column (this behaviour is probably expected).

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

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

4.3 The mono-row blocks

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

4.4 The mono-cell blocks

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

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

- It's possible to use the command `\\` in a (mono-cell) block.

- It's possible to use the option of horizontal alignment of the block in derogation of the type of column given in the preamble of the array.

- It's possible do draw a frame around the cell with the key `draw` of the command `\Block` and to fill the background with rounded corners with the keys `fill` and `rounded-corners`.[10]

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

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

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

Writer	year of birth
Hugo	1802
Balzac	1799

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

4.5 Horizontal position of the content of the block

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

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

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

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

5 The rules

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

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

5.1 Some differences with the classical environments

5.1.1 The vertical rules

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

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

First	Second
Peter	
Mary	George

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

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

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

a	b	c	d
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}}
```

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	B	C	D

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

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

A	B	C	D
A	B	C	D

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

5.2 The thickness and the color of the rules

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

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

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

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

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

5.3 The tools of nicematrix for the rules

Here are the tools provided by nicematrix for the rules.

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

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

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

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

5.3.1 The keys hlines and vlines

The keys `hlines` and `vlines` (which draw, of course, horizontal and vertical rules) take in as value a list of numbers which are the numbers of the rules to draw.[13]
In fact, for the environments with delimiters (such as `{pNiceMatrix}` or `{bNiceArray}`), the key `vlines` don't draw the exterior rules (this is certainly the expected behaviour).

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

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

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

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

5.3.2 The keys hvlines and hvlines-except-borders

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

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

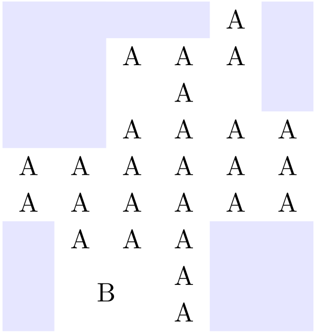
rose	tulipe	marguerite	dahlia
violette	fleurs		souci
pervenche			lys
arum	iris	jacinthe	muguet

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

5.3.3 The (empty) corners

The four `corners` of an array will be designed by `NW`, `SW`, `NE` and `SE` (*north west*, *south west*, *north east* and *south east*).
For each of these corners, we will call *empty corner* (or simply *corner*) the reunion of all the empty rectangles starting from the cell actually in the corner of the array.[14]
However, it's possible, for a cell without content, to require nicemarix to consider that cell as not empty with the key `\NotEmpty`.

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



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

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

```
\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 \\
  & \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		

[14] For sake of completeness, we should also say that a cell contained in a block (even an empty cell) is not taken into account for the determination of the corners. That behaviour is natural. The precise definition of a "non-empty cell" is given below (cf. p. 45).

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

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

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

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

5.4 The command \diagbox

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

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

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

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

5.5 Dotted rules

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

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

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

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

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

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

Remark: In the package array (on which the package nicematrix relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule[16]. In nicematrix, the dotted lines drawn by `\hdottedline` and "`:`" do likewise.

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

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

5.6 Commands for customized rules

New 6.5 It's possible to define commands and letters for customized rules with the key `custom-line` available in `\NiceMatrixOptions` and in the options of individual environments. That key takes in as argument a list of *key=value* pairs. First, there is two keys to define the tools which will be used to use that new type of rule.

- the key `command` is the name (without the backslahs) of a command that will be created by nicematrix and that will be available for the final user in order to draw horizontal rules (similarly to `\hline`);
- the key `letter` takes in as argument a letter[17] that the user will use in the preamble of an environment with preamble (such as `{NiceTabular}` in order to specify a vertical rule.

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

- *First possibility*

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

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

- *Second possibility*

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

- **New 6.6** *Third possibility*

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

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

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

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

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

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

Here is an example of the key `tikz`.

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

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

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

one	two	three
four	five	six
seven	eight	nine

6 The color of the rows and columns

6.1 Use of colortbl

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

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

However, there is two drawbacks:

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

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

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

The package nicematrix provides tools to avoid those problems.

6.2 The tools of nicematrix in the \CodeBefore

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

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

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

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

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

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

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

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

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

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

a	b	c
e	f	g
h	i	j

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

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

[19] Remark that, in the `\CodeBefore`, PGF/Tikz nodes of the form "`(i-|j)`" are also available to indicate the position to the potential rules: cf. p. 42.

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

a	b	c
e	f	g
h	i	j

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

- The command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

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

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

We have used the key `r` which aligns all the columns rightwards (cf. p. 36).

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

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

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

- The command `\columncolor` takes its name from the command `\columncolor` of colortbl. Its syntax is similar to the syntax of `\rowcolor`.

- The command `\rowcolors` (with a *s*) takes its name from the command `\rowcolors` of xcolor[20]. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows

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

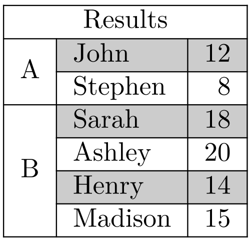
of the tabular with the tow colors (provided in second and third argument), beginning with the row whose number is given in first (mandatory) argument.

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`.

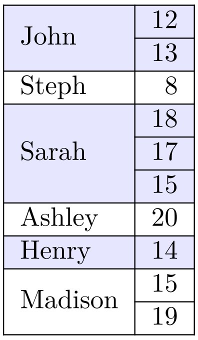
- The key `cols` describes a set of columns. The command `\rowcolors` will color only the cells of these columns. The value is a comma-separated list of intervals of the form *i*-*j* (where *i* or *j* may be replaced by `*`).
- With the key `restart`, each interval of rows (specified by the first mandatory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

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



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

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



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

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

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

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

Peter
James
Abigail
Elisabeth
Claudius
Jane
Alexandra

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

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

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

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

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

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

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

6.3 Color tools with the syntax of colortbl

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

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

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

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

7 The command \RowStyle

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

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

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

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

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

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

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

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

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

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

8 The width of the columns

8.1 Basic tools

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

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

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{array}{c}\begin{bmatrix} 9 & 17 \\ -2 & 5 \end{bmatrix} \\ \\ \begin{bmatrix} 1 & 1245345 \\ 345 & 2 \end{bmatrix}\end{array}$$

8.2 The columns V of varwidth

New 6.3

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

That point is illustrated on the following examples.

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

- first item
- second item

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

- first item
- second item

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

When the package varwidth is loaded, the columns V of varwidth are supported by nicematrix. Concerning nicematrix, one of the interests of this type of columns is that, for a cell of a column of type V, the PGF/Tikz node created by nicematrix for the content of that cell has a width adjusted to the content of the cell : cf. p. 40. If the content of the cell is empty, the cell will be considered as empty by nicematrix in the construction of the dotted lines and the «empty corners» (that's not the case with a cell of a column p, m or b).

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

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

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

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

8.3 The columns X

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

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

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

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

- It's possible to give a weight for the column by providing a positive integer directly as argument of the specifier `X`. For example, a column `X[2]` will have a width double of the width of a column `X` (which has a weight equal to 1).[28]

- It's possible to specify an horizontal alignment with one of the letters `l`, `c` and `r` (which insert respectively `\raggedright`, `\centering` and `\raggedleft` followed by `\arraybackslash`).

- It's possible to specify a vertical alignment with one of the keys `t` (alias `p`), `m` and `b` (which construct respectively columns of type `p`, `m` and `b`). The default value is `t`.

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

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

9 The exterior rows and columns

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

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

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

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

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

$$\begin{array}{ccc}
 & \begin{matrix} C_1 & \cdots & \cdots & C_4 \end{matrix} & \\
\begin{matrix} L_1 \\ \vdots \\ \vdots \\ L_4 \end{matrix} & \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} & \begin{matrix} L_1 \\ \vdots \\ \vdots \\ L_4 \end{matrix} \\
 & \begin{matrix} C_1 & \cdots & \cdots & C_4 \end{matrix} &
\end{array}$$

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

We have several remarks to do.

- For the environments with an explicit preamble (i.e. `{NiceTabular}`, `{NiceArray}` and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type `r` for the first column and `l` for the last one.[29]
- One may wonder how nicematrix determines the number of rows and columns which are needed for the composition of the "last row" and "last column".
  - For the environments with explicit preamble, like `{NiceTabular}` and `{pNiceArray}`, the number of columns can obviously be computed from the preamble.
  - When the option `light-syntax` (cf. p. 38) is used, nicematrix has, in any case, to load the whole body of the environment (and that's why it's not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (but not the number of columns).
  - In the other cases, nicematrix compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

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

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

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

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

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

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

Remarks

- As shown in the previous example, the horizontal and vertical rules don't extend in the exterior rows and columns. This remark also applies to the customized rules created by the key `custom-line` (cf. p. 12).
- A specification of color present in `code-for-first-row` also applies to a dotted line drawn in that exterior "first row" (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 19) doesn't apply to the "first column" and "last column".
- For technical reasons, it's not possible to use the option of the command `\\` after the "first row" or before the "last row". The placement of the delimiters would be wrong. If you are looking for a workaround, consider the command `\SubMatrix` in the `\CodeAfter` described p. 29.

10 The continuous dotted lines

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

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells[31] on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it's an horizontal line; for `\Vdots`, it's a vertical line and for `\Ddots` and `\Iddots` diagonal ones. It's possible to change the color of these lines with the option `color`.[32]

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

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

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}
```

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

[30] The 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.

[31] The precise definition of a "non-empty cell" is given below (cf. p. 45).

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

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

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

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

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

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

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

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

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

However, a command `\hspace*` might interfer 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 \\[1cm]
0      & \Cdots &               & 0
\end{bNiceMatrix}
```

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

10.1 The option nullify-dots

Consider the following matrix composed classicaly 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.

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

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

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

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

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

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

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

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

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

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

10.2 The commands \Hdotsfor and \Vdotsfor

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

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

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

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

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

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

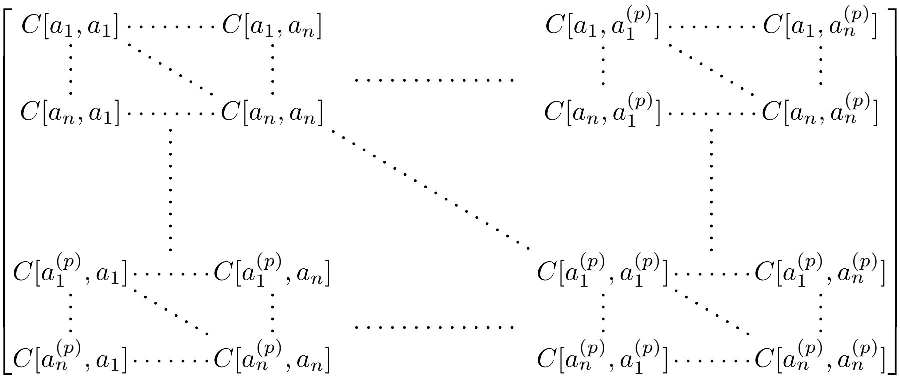
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ & \ldots & \ldots & \ldots & \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

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

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

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

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



$C[a_1,a_1]$ $C[a_1,a_n]$ $C[a_1,a_1^{(p)}]$ $C[a_1,a_n^{(p)}]$
$C[a_n,a_1]$ $C[a_n,a_n]$ $C[a_n,a_1^{(p)}]$ $C[a_n,a_n^{(p)}]$
$C[a_1^{(p)},a_1]$ $C[a_1^{(p)},a_n]$ $C[a_1^{(p)},a_1^{(p)}]$ $C[a_1^{(p)},a_n^{(p)}]$
$C[a_n^{(p)},a_1]$ $C[a_n^{(p)},a_n]$ $C[a_n^{(p)},a_1^{(p)}]$ $C[a_n^{(p)},a_n^{(p)}]$

10.3 How to generate the continuous dotted lines transparently

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

- The option `renew-dots`

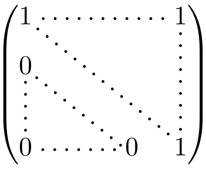
  With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`[30] and `\hdotsfor` are redefined within the environments provided by nicematrix and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` ("automatic dots" of `amsmath`) is also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

- The option `renew-matrix`

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

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

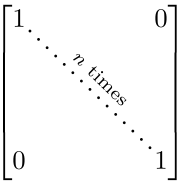
```
\NiceMatrixOptions{renew-dots,renew-matrix}
\begin{pmatrix}
1      & \cdots & \cdots & 1      \\
0      & \ddots &        & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
0      & \cdots & 0      & 1
\end{pmatrix}
```



10.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `\CodeAfter` which is described p. 28) accept two optional arguments specified by the tokens _ and ^ for labels positionned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

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



[35] The 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 conjonction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.

10.5 Customisation of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and `\Vdotsfor` (and by the command `\line` in the `\CodeAfter` which is described p. 28) may be customized by three options (specified between square brackets after the command):

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

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

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

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

The option xdots/color

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

The option xdots/shorten

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

The option xdots/line-style

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

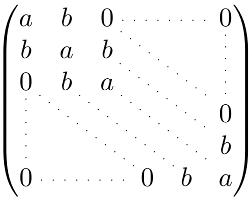
```
\tikz \draw [dotted] (0,0) -- (5,0) ;
```

In order to provide lines with rounded dots in the style of those provided by `\ldots` (at least with the *Computer Modern* fonts), the package nicematrix embeds its own system to draw a dotted line (and this system uses PGF and not Tikz). This style is called `standard` and that's the initial value of the parameter `xdots/line-style`.
However (when Tikz is loaded) it's possible to use for `xdots/line-style` any style provided by Tikz, that is to say any sequence of options provided by Tikz for the Tizk pathes (with the exception of "`color`", "`shorten >`" and "`shorten <`").

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

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

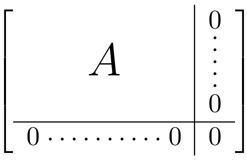
[36] The first reason of this behaviour is that the PDF format includes a description for dashed lines. The lines specified with this descriptor are displayed very efficiently by the PDF readers. It's easy, starting from these dashed lines, to create a line composed by square dots whereas a line of rounded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



10.6 The dotted lines and the rules

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

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



11 The \CodeAfter

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

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

The experienced users may, for instance, use the PGF/Tikz nodes created by nicematrix in the `\CodeAfter`. These nodes are described further beginning on p. 39.

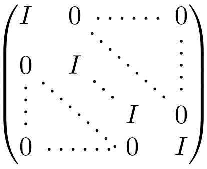
Moreover, several special commands are available in the `\CodeAfter`: `line`, `\SubMatrix`, `\OverBrace` and `\UnderBrace`. We will now present these commands.

11.1 The command \line in the \CodeAfter

The command `\line` draws directly dotted lines between nodes. It takes in two arguments for the two cells to link, both of the form i-j where is the number of the row and j is the number of the column. The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 27).

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

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

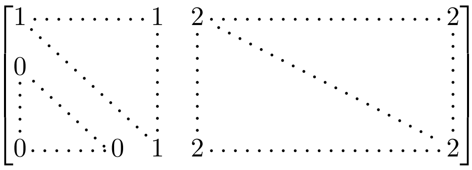


[37] On the other side, the command `\line` in the `\CodeAfter` (cf. p. 28) does *not* create block.

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

It can also be used to draw a diagonal line not parallel to the other diagonal lines (by default, the dotted lines drawn by `\Ddots` are "parallelized": cf. p. 45).

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



11.2 The command \SubMatrix in the \CodeAfter

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

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

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

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

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

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

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

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \left[\begin{matrix} a & b \\ c & d \end{matrix}\right]^{T} \end{bmatrix}$$

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

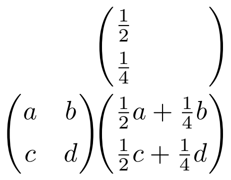
[39] There is no optional argument between square brackets in first position because a square bracket just after `\SubMatrix` must be interpreted as the first (mandatory) argument of the command `\SubMatrix`: that bracket is the left delimiter of the sub-matrix to construct (eg.: `\SubMatrix[{2-2}{4-7}]`).

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

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

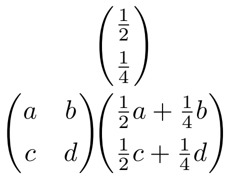
All these keys are also available in `\NiceMatrixOptions`, at the level of the environments of nicematrix or as option of the command `\CodeAfter` with the prefix `sub-matrix` which means that their names are therefore `sub-matrix/left-xshift`, `sub-matrix/right-xshift`, `sub-matrix/xshift`, etc.

```
$\begin{NiceArray}{cc@{\hspace{5mm}}l}[cell-space-limits=2pt]
  &   & \frac12 \\
  &   & \frac14 \\[1mm]
a & b & \frac12a+\frac14b \\
c & d & \frac12c+\frac14d \\
\CodeAfter
  \SubMatrix({1-3}{2-3})
  \SubMatrix({3-1}{4-2})
  \SubMatrix({3-3}{4-3})
\end{NiceArray}$
```



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

```
$\begin{NiceArray}{cc@{\hspace{5mm}}l}[cell-space-limits=2pt]
  &   & \frac12 \\
  &   & \frac14 \\[1mm]
a & b & \frac12a+\frac14b \\
c & d & \frac12c+\frac14d \\
\CodeAfter
  \SubMatrix({1-3}{2-3})[slim]
  \SubMatrix({3-1}{4-2})
  \SubMatrix({3-3}{4-3})
\end{NiceArray}$
```



There is also a key `name` which gives a name to the submatrix created by `\SubMatrix`. That name is used to create PGF/Tikz nodes: cf p. 43.

It's also possible to specify some delimiters[40] 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.

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

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

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

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

New 6.4

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

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

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

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

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

- `left-shorten` and `right-shorten` which do not take in value; when the key `left-shorten` is used, the abscissa of the left extremity of the brace is computed with the contents of the cells of the involved sub-array, otherwise, the position of the potential vertical rule is used (idem for `right-shorten`).
- `shorten`, which is the conjunction of the keys `left-shorten` and `right-shorten`;
- `yshift`, which shifts vertically the brace (and its label) ;
- **New 6.7** `color`, which sets the color of the brace (and its label).

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

$$\left(\overbrace{\begin{matrix}1&2&3\\11&12&13\end{matrix}}^{A}\;\overbrace{\begin{matrix}4&5&6\\14&15&16\end{matrix}}^{B}\right)$$

12 The notes in the tabulars

12.1 The footnotes

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

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

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

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

12.2 The notes of tabular

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

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

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

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

[a] Achard is an old family of the Poitou.

[b] The name Lefebvre is an alteration of the name Lefebure.

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

- If a command `\tabularnote{...}` is exactly at the end of a cell (with no space at all after), the label of the note is composed in an overlapping position (towards the right). This structure may provide a better alignment of the cells of a given column.

- If the key `notes/para` is used, the notes are composed at the end of the array in a single paragraph (as with the key `para` of threeparttable).

- There is a key `tabularnote` which provides a way to insert some text in the zone of the notes before the numbered tabular notes.

- If the package booktabs has been loaded (before or after nicematrix), the key `notes/bottomrule` draws a `\bottomrule` of booktabs *after* the notes.

- The command `\tabularnote` may be used *before* the environment of nicematrix. Thus, it's possible to use it on the title inserted by `\caption` in an environment `{table}` of LaTeX.

- It's possible to create a reference to a tabular note created by `\tabularnote` (with the usual command `\label` used after the `\tabularnote`).

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

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

Table 1: Use of `\tabularnote`[a]

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

Some text before the notes.

[a] It's possible to put a note in the caption.
[b] Considered as the first nurse of history.
[c] Nicknamed “the Lady with the Lamp”.
[d] The label of the note is overlapping.

12.3 Customisation of the tabular notes

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

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

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

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

We detail these keys.

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

  Initial value: `false`

  That key is also available within a given environment.

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

  Initial value: `false`

  That key is also available within a given environment.

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

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

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

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

  Initial value: `\textsuperscript{#1}`

  In French, it's a tradition of putting a small space before the label of note. That tuning could be acheived 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: `\textsuperscript{#1}`

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

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

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

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

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

  The specification `align = left` in that style requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of booktabs (see for example the table 1, p. 33).

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

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

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

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

  Initial value: *empty*

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

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

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

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

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

12.4 Use of {NiceTabular} with threeparttable

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

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

13 Other features

13.1 Use of the column type S of siunitx

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

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

$$\begin{array}{cccc} C_1 & \cdots & \cdots & C_n \\ \left(\begin{matrix} 2.3 \\ 12.4 \\ 1.45 \\ 7.2 \end{matrix}\right. & \begin{matrix} 0 \\ \vdots \\ \vdots \\ 0 \end{matrix} & \begin{matrix} \cdots \\ \\ \\ \cdots \end{matrix} & \left.\begin{matrix} 0 \\ \vdots \\ \vdots \\ 0 \end{matrix}\right) \end{array}$$

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

13.2 Alignment option in {NiceMatrix}

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

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

$$\left[\begin{array}{rr} \cos x & -\sin x \\ \sin x & \cos x \end{array}\right]$$

13.3 The command \rotate

The package nicematrix provides a command `\rotate`. When used in the beginning of a cell, this command composes the contents of the cell after a rotation of 90° in the direct sens.
In the following command, we use that command in the `code-for-first-row`.[41]

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

$$A = \begin{array}{c} \begin{matrix} \rotatebox{90}{$\scriptstyle\text{image of } e_1$} & \rotatebox{90}{$\scriptstyle\text{image of } e_2$} & \rotatebox{90}{$\scriptstyle\text{image of } e_3$} & \end{matrix} \\ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{matrix} \scriptstyle e_1 \\ \scriptstyle e_2 \\ \scriptstyle e_3 \end{matrix} \end{array}$$

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

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

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

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

13.4 The option small

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

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

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

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

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

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

13.5 The counters iRow and jCol

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column[42]. Of course, the user must not change the value of these counters which are used internally by nicematrix.
In the `\CodeBefore` (cf. p. 14) and in the `\CodeAfter` (cf. p. 28), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

[42] We recall that the exterior "first row" (if it exists) has the number 0 and that the exterior "first column" (if it exists) has also the number 0.

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

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

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

The package nicematrix also provides commands in order to compose automatically matrices from a general pattern. These commands are `\AutoNiceMatrix`, `\pAutoNiceMatrix`, `\bAutoNiceMatrix`, `\vAutoNiceMatrix`, `\VAutoNiceMatrix` and `\BAutoNiceMatrix`.
These commands take in two mandatory arguments. The first is the format of the matrix, with the syntax n-p where n is the number of rows and p the number of columns. The second argument is the pattern (it's a list of tokens which are inserted in each cell of the constructed matrix).

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

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

13.6 The option light-syntax

The option `light-syntax` (inpired by the package spalign) allows the user to compose the arrays with a lighter syntax, which gives a better legibility of the TeX source.
When this option is used, one should use the semicolon for the end of a row and spaces or tabulations to separate the columns. However, as usual in the TeX world, the spaces after a control sequence are discarded and the elements between curly braces are considered as a whole.

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

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

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

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

13.7 Color of the delimiters

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

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

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

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

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

13.8 The environment {NiceArrayWithDelims}

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

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

$$\left\downarrow\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}\right\uparrow$$

13.9 The command \OnlyMainNiceMatrix

The command `\OnlyMainNiceMatrix` executes its argument only when it is in the main part of the array, that is to say it is not in one of the exterior rows. If it is used outside an environment of nicematrix, that command is no-op.
For an example of utilisation, see `tex.stackexchange.com/questions/488566`

14 Use of Tikz with nicematrix

14.1 The nodes corresponding to the contents of the cells

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

14.1.1 The columns V of varwidth

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

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

Titre

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

We have used the nodes corresponding to the position of the potential rules, which are described below (cf. p. 42).

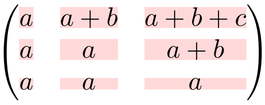
14.2 The "medium nodes" and the "large nodes"

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

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

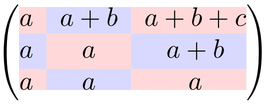
[45] There is also an option `create-extra-nodes` which is an alias for the conjonction of `create-medium-nodes` and `create-large-nodes`.

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



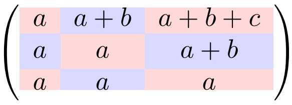
$$\begin{pmatrix} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{pmatrix}$$

The names of the “large nodes” are constructed by adding the suffix “`-large`” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.[46]



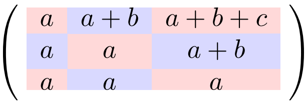
$$\begin{pmatrix} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{pmatrix}$$

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



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

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



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

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

Here is an array composed with the following code:

```
\large
\begin{NiceTabular}{wl{2cm}ll}[hvlines]
fraise & amande & abricot \\
prune & pêche & poire  \\[1ex]
noix & noisette & brugnon
\end{NiceTabular}
```

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

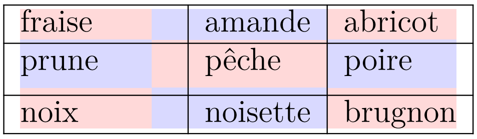
Here, we have colored all the cells of the array with `\chessboardcolors`.

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

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

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

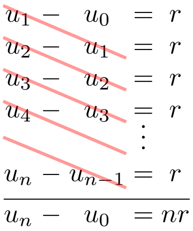


fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

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

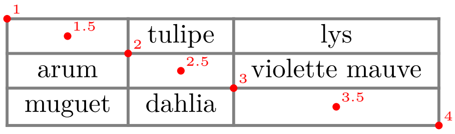
```
\begin{NiceArray}{c@{\;}c@{\;}c@{\;}c@{\;}c}[create-medium-nodes]
    u_1 &-& u_0 &=& r     \\
    u_2 &-& u_1 &=& r     \\
    u_3 &-& u_2 &=& r     \\
    u_4 &-& u_3 &=& r     \\
    \phantom{u_5} & &  \phantom{u_4}    &\smash{\vdots} &       \\
    u_n &-& u_{n-1} &=& r \\[3pt]
    \hline
    u_n &-& u_0 &=& nr \\
\CodeAfter
    \tikz[very thick, red, opacity=0.4,name suffix = -medium]
    \draw (1-1.north west) -- (2-3.south east)
    (2-1.north west) -- (3-3.south east)
    (3-1.north west) -- (4-3.south east)
    (4-1.north west) -- (5-3.south east)
    (5-1.north west) -- (6-3.south east) ;
\end{NiceArray}
```



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

14.3 The nodes which indicate the position of the rules

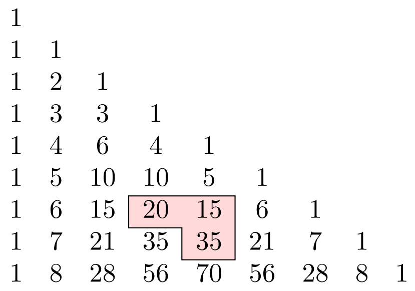
The package nicematrix creates a PGF/Tikz node merely called i (with the classical prefix) at the intersection of the horizontal rule of number i and the vertical rule of number i (more specifically the potential position of those rules because maybe there are not actually drawn). The last node has also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



	tulipe	lys
arum		violette mauve
muguet	dahlia	

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

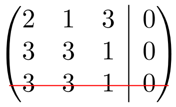
```
\begin{NiceMatrix}
\CodeBefore
  \tikz \draw [fill=red!15] (7-|4) |- (8-|5) |- (9-|6) |- cycle ;
\Body
1 \\
1 & 1 \\
1 & 2 &  1 \\
1 & 3 &  3 &  1 \\
1 & 4 &  6 &  4 &  1 \\
1 & 5 & 10 & 10 &  5 &  1 \\
1 & 6 & 15 & 20 & 15 &  6 &  1 \\
1 & 7 & 21 & 35 & 35 & 21 &  7 & 1 \\
1 & 8 & 28 & 56 & 70 &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



$$\begin{matrix}
1 \\
1 & 1 \\
1 & 2 & 1 \\
1 & 3 & 3 & 1 \\
1 & 4 & 6 & 4 & 1 \\
1 & 5 & 10 & 10 & 5 & 1 \\
1 & 6 & 15 & 20 & 15 & 6 & 1 \\
1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \\
1 & 8 & 28 & 56 & 70 & 56 & 28 & 8 & 1
\end{matrix}$$

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

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



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

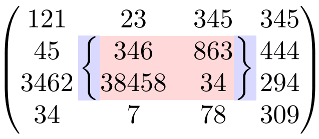
14.4 The nodes corresponding to the command \SubMatrix

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

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

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

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



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

15 API for the developpers

The package nicematrix provides two variables which are internal but public[48]:

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

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

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

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

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

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

Here is an example of utilisation:

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

merlan	requin	cabillaud
baleine	╳	morue
mante	raie	poule

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

16 Technical remarks

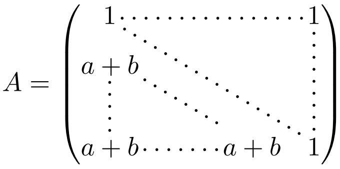
16.1 Diagonal lines

By default, all the diagonal lines[49] of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That’s why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

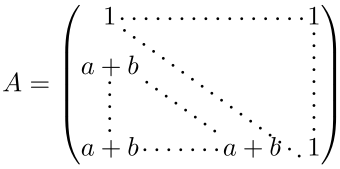
In the following examples, the first `\Ddots` instruction is written in color:

Example with parallelization (default):

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

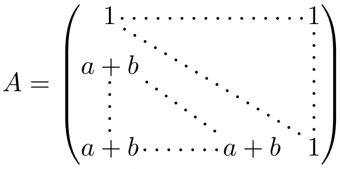


```
$A = \begin{pNiceMatrix}
1      & \Cdots &        & 1      \\
a+b    &        &        & \Vdots \\
\Vdots & \Ddots & \Ddots &        \\
a+b    & \Cdots & a+b    & 1
\end{pNiceMatrix}$
```



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

The same example without parallelization:



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

16.2 The “empty” cells

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

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

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

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

- Each cell whose TeX ouput has a width equal to zero is empty.

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

- A cell containing the command \NotEmpty is not empty (and a PGF/Tikz node) is created in that cell.

- A cell with a command \Hspace (or \Hspace*) is empty. This command \Hspace is a command defined by the package nicematrix with the same meaning as \hspace except that the cell where it is used is considered as empty. This command can be used to fix the width of some columns of the matrix without interfering with nicematrix.

- A cell of a column of type p, m or t is always considered as not empty. *Caution* : One should not rely upon that point because it may change in a future version of nicematrix. On the other side, a cell of a column of type V of varwidth (cf. p. 20) is empty when its TeX content has a width equal to zero.

16.3 The option exterior-arraycolsep

The environment {array} inserts an horizontal space equal to \arraycolsep before and after each column. In particular, there is a space equal to \arraycolsep before and after the array. This feature of the environment {array} was probably not a good idea[50]. The environment {matrix} of amsmath and its variants ({pmatrix}, {vmatrix}, etc.) of amsmath prefer to delete these spaces with explicit instructions \hskip -\arraycolsep[51]. The package nicematrix does the same in all its environments, {NiceArray} included. However, if the user wants the environment {NiceArray} behaving by default like the environment {array} of array (for example, when adapting an existing document) it's possible to control this behaviour with the option exterior-arraycolsep, set by the command \NiceMatrixOptions. With this option, exterior spaces of length \arraycolsep will be inserted in the environments {NiceArray} (the other environments of nicematrix are not affected).

16.4 Incompatibilities

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

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

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

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

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

The package nicematrix is not fully compatible with the package arydshln (because this package redefines many internal of array). By any means, in the context of nicematrix, it's recommended to draw dashed rules with the tools provided by nicematrix, by creating a customized line style with custom-line: cf. p. 12.

[50] In 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).*

[51] And 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.

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

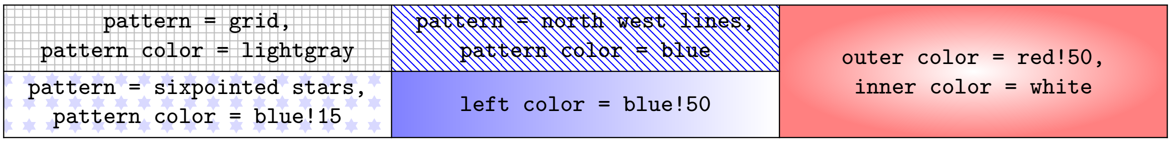
17 Examples

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

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

```
\usetikzlibrary{patterns}
```

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



<table>
<tr><td>pattern = grid,
pattern color = lightgray</td><td>pattern = north west lines,
pattern color = blue</td><td rowspan="2">outer color = red!50,
inner color = white</td></tr>
<tr><td>pattern = sixpointed stars,
pattern color = blue!15</td><td>left color = blue!50</td></tr>
</table>

17.2 Notes in the tabulars

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

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

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

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

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

[53] By default, nicematrix only loads PGF, which is a sub-layer of Tikz.
[54] Of course, it’s realistic only when there is very few notes in the tabular.
[55] In fact: the value of its argument.

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

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

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

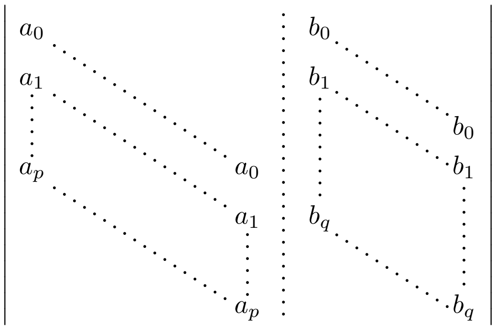
*Achard is an old family of the Poitou.

**The name Lefebvre is an alteration of the name Lefebure.

17.3 Dotted lines

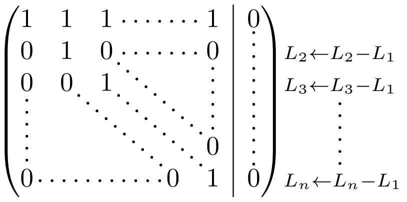
An example with the resultant of two polynoms:

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



An example for a linear system:

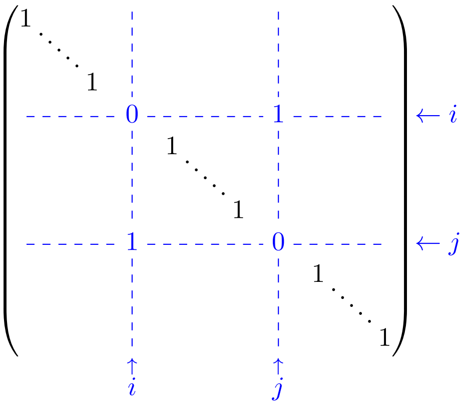
```
$\begin{pNiceArray}{*6c|c}[nullify-dots,last-col,code-for-last-col=\scriptstyle]
1      & 1 & 1 &\Cdots &   & 1      & 0      & \\
0      & 1 & 0 &\Cdots &   & 0      &        & L_2 \gets L_2-L_1 \\
0      & 0 & 1 &\Ddots &   & \Vdots &        & L_3 \gets L_3-L_1 \\
       &   &   &\Ddots &   & 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


1
1
1
1
0
0
1
0
0
0
0
1
0
0
1
0
0
$L_2 \leftarrow L_2 - L_1$
$L_3 \leftarrow L_3 - L_1$
$L_n \leftarrow L_n - L_1$

17.4 Dotted lines which are no longer dotted

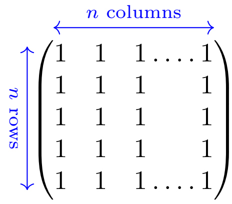
The option `line-style` controls the style of the lines drawn by `\Ldots`, `\Cdots`, etc. Thus, it's possible with these commands to draw lines which are not longer dotted.

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



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

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



17.5 Dashed rules

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

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

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

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 4 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{pmatrix}$$

17.6 Stacks of matrices

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

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

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

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

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

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

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

\end{NiceMatrixBlock}
```

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

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

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

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

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

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

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

...
\end{NiceMatrixBlock}
```

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

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

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

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

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

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

$$\begin{array}{l}
\left(\begin{array}{rrrr|r}12&-8&7&5&3\\3&-18&12&1&4\\-3&-46&29&-2&-15\\9&10&-5&4&7\end{array}\right)\\
\left(\begin{array}{rrrr|r}12&-8&7&5&3\\0&64&-41&1&19\\0&-192&123&-3&-57\\0&-64&41&-1&-19\end{array}\right)\begin{array}{l}\\ \scriptstyle L_2\gets L_1-4L_2\\ \scriptstyle L_3\gets L_1+4L_3\\ \scriptstyle L_4\gets 3L_1-4L_4\end{array}\\
\left(\begin{array}{rrrr|r}12&-8&7&5&3\\0&64&-41&1&19\\0&0&0&0&0\end{array}\right)\begin{array}{l}\\ \\ \scriptstyle L_3\gets 3L_2+L_3\end{array}\\
\left(\begin{array}{rrrr|r}12&-8&7&5&3\\0&64&-41&1&19\end{array}\right)
\end{array}$$

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

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

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

$$\begin{array}{l}
\left(\begin{array}{rrrr|r} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{array}\right) \\
\left(\begin{array}{rrrr|r} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{array}\right) \begin{array}{l} \\ L_2 \gets L_1-4L_2 \\ L_3 \gets L_1+4L_3 \\ L_4 \gets 3L_1-4L_4 \end{array} \\
\left(\begin{array}{rrrr|r} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right) \begin{array}{l} \\ \\ L_3 \gets 3L_2+L_3 \end{array} \\
\left(\begin{array}{rrrr|r} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \end{array}\right)
\end{array}$$

17.7 How to highlight cells of a matrix

In order to highlight a cell of a matrix, it's possible to "draw" that cell with the key `draw` of the command `\Block` (this is one of the uses of a mono-cell block[57]).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

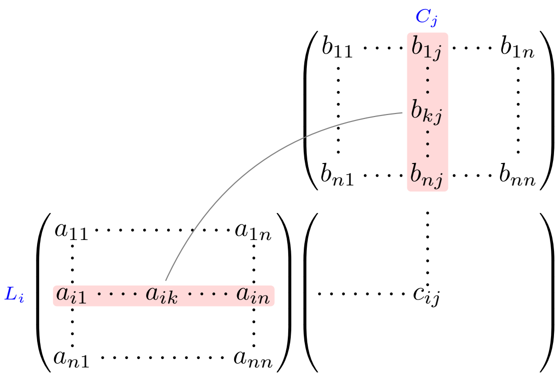
```
\[\begin{pNiceArray}{ccc}[last-col,create-medium-nodes]
\CodeBefore [create-cell-nodes]
  \begin{tikzpicture} [name suffix = -medium]
  \node [highlight = (1-1) (1-3)] {} ;
  \node [highlight = (2-1) (2-3)] {} ;
  \node [highlight = (3-1) (3-3)] {} ;
  \end{tikzpicture}
\Body
a & a + b & a + b +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

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

17.8 Utilisation of \SubMatrix in the \CodeBefore

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

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



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

```
\[\begin{NiceArray}{*{6}{c}@{\hspace{6mm}}*{5}{c}}[nullify-dots]
\CodeBefore [create-cell-nodes]
  \SubMatrix({2-7}{6-11})
  \SubMatrix({7-2}{11-6})
  \SubMatrix({7-7}{11-11})
  \begin{tikzpicture}
    \node [highlight = (9-2) (9-6)] { } ;
    \node [highlight = (2-9) (6-9)] { } ;
  \end{tikzpicture}
\Body
    &        &        &        &        &        &        & \color{blue}\scriptstyle C_j \\
    &        &        &        &        & b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\
    &        &        &        &        & \Vdots &        & \Vdots &        & \Vdots \\
    &        &        &        &        &        &        & b_{kj} \\
    &        &        &        &        &        &        & \Vdots \\
    &        &        &        &        & b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn} \\[3mm]
    & a_{11} & \Cdots &        &        & a_{1n} \\
    & \Vdots &        &        & \Vdots &        &        & \Vdots \\
\color{blue}\scriptstyle L_i
    & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} & \Cdots &        & c_{ij} \\
    & \Vdots &        &        & \Vdots \\
    & a_{n1} & \Cdots &        &        & a_{nn}  \\
\CodeAfter
\tikz \draw [gray,shorten > = 1mm, shorten < = 1mm] (9-4.north) to [bend left] (4-9.west) ;
\end{NiceArray}\]
```

18 Implementation

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

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

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

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

Declaration of the package and packages loaded

The prefix `nicematrix` has been registred 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`.

```
\RequirePackage{pgfcore}
\usepgfmodule{shapes}
```

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

```
\RequirePackage{l3keys2e}
\ProvidesExplPackage
  {nicematrix}
  {\myfiledate}
  {\myfileversion}
  {Enhanced arrays with the help of PGF/TikZ}
```

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

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

```
\RequirePackage { xparse }
\RequirePackage { array }
\RequirePackage { amsmath }
```

```
\cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
\cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
\cs_generate_variant:Nn \@@_error:nn { n x }
\cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
\cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
\cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_new:nnn { \msg_new:nnnn { nicematrix } }
```

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

Technical definitions

```
\tl_new:N \l_@@_argspec_tl
```

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

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \@ifpackageloaded { varwidth }
      { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_true_bool } }
      { \bool_const:Nn \c_@@_varwidth_loaded_bool { \c_false_bool } }
    \@ifpackageloaded { arydshln }
      { \bool_const:Nn \c_@@_arydshln_loaded_bool { \c_true_bool } }
      { \bool_const:Nn \c_@@_arydshln_loaded_bool { \c_false_bool } }
    \@ifpackageloaded { booktabs }
      { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_true_bool } }
      { \bool_const:Nn \c_@@_booktabs_loaded_bool { \c_false_bool } }
    \@ifpackageloaded { enumitem }
      { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_true_bool } }
      { \bool_const:Nn \c_@@_enumitem_loaded_bool { \c_false_bool } }
    \@ifpackageloaded { tabularx }
      { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_true_bool } }
      { \bool_const:Nn \c_@@_tabularx_loaded_bool { \c_false_bool } }
      { }
    \@ifpackageloaded { tikz }
      {
```

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-\endtikpicture (or \begin{tikzpicture}-\end{tikzpicture}) must be statically "visible" (even when externalization is not activated).
That's why we create \c_@@_pgfortikzpicture_tl and \c_@@_endpgfortikzpicture_tl which will be used to construct in a \AtBeginDocument the correct version of some commands. The tokens \exp_not:N are mandatory.

```
        \bool_const:Nn \c_@@_tikz_loaded_bool \c_true_bool
        \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
        \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
      }
      {
        \bool_const:Nn \c_@@_tikz_loaded_bool \c_false_bool
        \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
        \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
      }
  }
```

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 programmation. At the date January 2022, the current version revtex4-2 is 4.2e (compatible with booktabs).

```
\@ifclassloaded { revtex4-1 }
  { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
  {
    \@ifclassloaded { revtex4-2 }
      { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
      {
```

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

```
        \cs_if_exist:NT \rvtx@ifformat@geq
          { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
          { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
      }
  }
```

```
\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.

```
\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.

```
\cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
  {
    \iow_now:Nn \@mainaux
      {
        \ExplSyntaxOn
        \cs_if_free:NT \pgfsyspdfmark
          { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
        \ExplSyntaxOff
      }
    \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
  }
```

We define a command \iddots similar to \ddots (⋱) but with dots going forward (⋰). 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.

```
\ProvideDocumentCommand \iddots { }
  {
    \mathinner
      {
        \tex_mkern:D 1 mu
        \box_move_up:nn { 1 pt } { \hbox:n { . } }
        \tex_mkern:D 2 mu
        \box_move_up:nn { 4 pt } { \hbox:n { . } }
        \tex_mkern:D 2 mu
        \box_move_up:nn { 7 pt }
          { \vbox:n { \kern 7 pt \hbox:n { . } } }
        \tex_mkern:D 1 mu
      }
  }
```

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.

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \@ifpackageloaded { booktabs }
      { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
      { }
  }
\cs_set_protected:Npn \nicematrix@redefine@check@rerun
  {
    \cs_set_eq:NN \@@_old_pgfutil@check@rerun \pgfutil@check@rerun
```

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

```
    \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
      {
        \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
          { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
      }
  }
```

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

```
\bool_new:N \l_@@_colortbl_loaded_bool
\hook_gput_code:nnn { begindocument } { . }
  {
    \@ifpackageloaded { colortbl }
      { \bool_set_true:N \l_@@_colortbl_loaded_bool }
      {
```

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.

```
        \cs_set_protected:Npn \CT@arc@ { }
        \cs_set:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
        \cs_set:Npn \CT@arc #1 #2
          {
            \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
              { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
          }
```

Idem for \CT@drs@.

```
        \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
        \cs_set:Npn\CT@drs #1 #2
          {
            \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
              { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
          }
        \cs_set:Npn \hline
          {
            \noalign { \ifnum 0 = `} \fi
            \cs_set_eq:NN \hskip \vskip
            \cs_set_eq:NN \vrule \hrule
            \cs_set_eq:NN \@width \@height
            { \CT@arc@ \vline }
            \futurelet \reserved@a
            \@xhline
          }
      }
  }
```

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.

```
\cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
\cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
  {
    \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
    \int_compare:nNnT { #1 } > 1 { \multispan { \int_eval:n { #1 - 1 } } & }
    \multispan { \int_eval:n { #2 - #1 + 1 } }
    {
      \CT@arc@
      \leaders \hrule \@height \arrayrulewidth \hfill
```

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

```
      \skip_horizontal:N \c_zero_dim
    }
```

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`.

```
    \everycr { }
    \cr
    \noalign { \skip_vertical:N -\arrayrulewidth }
  }
```

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.

```
\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`.

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

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

```
\cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
\cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
  {
    \tl_if_empty:nTF { #3 }
      { \@@_cline_iii:w #1|#2-#2 \q_stop }
      { \@@_cline_ii:w #1|#2-#3 \q_stop }
  }
\cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
  { \@@_cline_iii:w #1|#2-#3 \q_stop }
\cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
  {
```

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).

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

[59] See question 99041 on TeX StackExchange.

```
      {
        \CT@arc@
        \leaders \hrule \@height \arrayrulewidth \hfill
        \skip_horizontal:N \c_zero_dim
      }
```

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

```
    \peek_meaning_remove_ignore_spaces:NTF \cline
      { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
      { \everycr { } \cr }
  }
\cs_generate_variant:Nn \@@_cline_i:nn { e n }
```

The following command is a small shortcut.

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

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

```
\cs_new_protected:Npn \@@_set_CT@drsc@:
  { \peek_meaning:NTF [ \@@_set_CT@drsc@_i: \@@_set_CT@drsc@_ii: }
\cs_new_protected:Npn \@@_set_CT@drsc@_i: [ #1 ] #2 \q_stop
  { \cs_set:Npn \CT@drsc@ { \color [ #1 ] { #2 } } }
\cs_new_protected:Npn \@@_set_CT@drsc@_ii: #1 \q_stop
  { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
```

```
\cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor
```

The column S of siunitx

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

```
\bool_new:N \l_@@_siunitx_loaded_bool
\hook_gput_code:nnn { begindocument } { . }
  {
    \@ifpackageloaded { siunitx }
      { \bool_set_true:N \l_@@_siunitx_loaded_bool }
      { }
  }
```

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

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \bool_if:nTF { ! \l_@@_siunitx_loaded_bool }
      { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
      {
        \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
          {
            \renewcommand*{\NC@rewrite@S}[1][]
              {
```

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

```
                \@temptokena \exp_after:wN
                  { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
                \NC@find
              }
          }
```

```
      }
  }
```

Parameters

The following counter will count the environments {NiceArray}. The value of this counter will be used to prefix the names of the Tikz nodes created in the array.

```
\int_new:N \g_@@_env_int
```

The following command is only a syntaxic shortcut. It must *not* be protected (it will be used in names of PGF nodes).

```
\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.

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

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

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

The following counter will count the environments {NiceMatrixBlock}.

```
\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).

```
\dim_new:N \l_@@_columns_width_dim
```

The dimension `\l_@@_col_width_dim` will be available in each cell which belongs to a column of fixed width: `w{...}{...}`, `W{...}{...}`, `p{}`, `m{}`, `b{}` but also `X` (when the actual width of that column is known, that is to say after the first compilation). It's the width of that column. It will be used by some commands `\Block`. A non positive value means that the column has no fixed width (it's a column of type `c`, `r`, `l`, etc.).

```
\dim_new:N \l_@@_col_width_dim
\dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

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

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

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

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

```
\str_new:N \l_@@_hpos_cell_str
\str_set:Nn \l_@@_hpos_cell_str { c }
```

When there is a mono-column block (created by the command \Block), we want to take into account the width of that block for the width of the column. That's why we compute the width of that block in the \g_@@_blocks_wd_dim and, after the construction of the box \l_@@_cell_box, we change the width of that box to take into account the length \g_@@_blocks_wd_dim.

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

Idem pour the mono-row blocks.

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

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

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

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

```
\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.

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

The following key corresponds to the key notes/detect_duplicates.

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

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

```
\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.

```
\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.

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

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

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

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

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

In a cell, it will be possible to know whether we are in a cell of a column of type X thanks to that flag.

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

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

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

```
\cs_new_protected:Npn \@@_test_if_math_mode:
  {
    \if_mode_math: \else:
      \@@_fatal:n { Outside~math~mode }
    \fi:
  }
```

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

```
\tl_new:N \l_@@_letter_vlism_tl
```

The list of the columns where vertical lines in sub-matrices (vlism) must be drawn. Of course, the actual value of this sequence will be known after the analyse of the preamble of the array.

```
\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”.

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

```
\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*.

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

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

```
\cs_new:Npn \@@_full_name_env:
  {
    \str_if_eq:VnTF \g_@@_com_or_env_str { command }
      { command \space \c_backslash_str \g_@@_name_env_str }
      { environment \space \{ \g_@@_name_env_str \} }
  }
```

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

```
\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.

```
\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.

```
\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.

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

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

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

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

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

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

The sum of the weights of all the `X`-columns in the preamble. The weight of a `X`-column is given as optional argument between square brackets. The default value, of course, is 1.

```
\int_new:N \g_@@_total_X_weight_int
```

If there is at least one `X`-column in the preamble of the array, the following flag will be raised via the `aux` file. The length `l_@@_x_columns_dim` will be the width of `X`-columns of weight 1 (the width of a column of weigth n will be that dimension multiplied by n). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
\bool_new:N \l_@@_X_columns_aux_bool
\dim_new:N \l_@@_X_columns_dim
```

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if `\Hdotsfor` is used in that column.

```
\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).

```
\bool_new:N \g_@@_row_of_col_done_bool
```

It's possible to use the command `\NotEmpty` to specify explicitely that a cell must be considered as non empty by `nicematrix` (the Tikz nodes are constructed only in the non empty cells).

```
\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`).

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

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

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

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

```
\dim_new:N \l_@@_x_initial_dim
\dim_new:N \l_@@_y_initial_dim
\dim_new:N \l_@@_x_final_dim
\dim_new:N \l_@@_y_final_dim
```

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

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

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

```
\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".

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

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

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

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

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

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

```
\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.

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

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

```
\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`).

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

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

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

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

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

```
\int_new:N \l_@@_row_min_int
\int_new:N \l_@@_row_max_int
\int_new:N \l_@@_col_min_int
\int_new:N \l_@@_col_max_int
```

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

```
\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).

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

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

```
\tl_new:N \l_@@_fill_tl
\tl_new:N \l_@@_draw_tl
\seq_new:N \l_@@_tikz_seq
\clist_new:N \l_@@_borders_clist
\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`.

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

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

```
\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`).

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

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

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

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

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

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

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

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

\dim_new:N \l_@@_submatrix_extra_height_dim
\dim_new:N \l_@@_submatrix_left_xshift_dim
\dim_new:N \l_@@_submatrix_right_xshift_dim
\clist_new:N \l_@@_hlines_clist
\clist_new:N \l_@@_vlines_clist
\clist_new:N \l_@@_submatrix_hlines_clist
\clist_new:N \l_@@_submatrix_vlines_clist
```

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

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

Variables for the exterior rows and columns

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

- **First row**

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

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

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

  ```
  \int_new:N \l_@@_last_row_int
  \int_set:Nn \l_@@_last_row_int { -2 }
  ```

  If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the “last row”.[60]

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

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

Idem for `\l_@@_last_col_without_value_bool`

```
\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.

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

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

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

Some utilities

```
\cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
  {
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_set:Nn \l_tmpb_tl { #2 }
  }
```

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

```
\cs_new_protected:Npn \@@_expand_clist:N #1
  {
    \clist_if_in:NnF #1 { all }
      {
        \clist_clear:N \l_tmpa_clist
        \clist_map_inline:Nn #1
          {
            \tl_if_in:nnTF { ##1 } { - }
              { \@@_cut_on_hyphen:w ##1 \q_stop }
              {
                \tl_set:Nn \l_tmpa_tl { ##1 }
                \tl_set:Nn \l_tmpb_tl { ##1 }
              }
            \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
              { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
          }
        \tl_set_eq:NN #1 \l_tmpa_clist
      }
  }
```

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.

```
\newcounter { tabularnote }
```

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

```
\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.

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

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

```
\newcounter{nicematrix_draft}
\cs_new_protected:Npn \@@_notes_format:n #1
  {
    \setcounter { nicematrix_draft } { #1 }
    \@@_notes_style:n { nicematrix_draft }
  }
```

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

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

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

```
\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`.

```
\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`.

```
\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).

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
      {
        \NewDocumentCommand \tabularnote { m }
          { \@@_error:n { enumitem~not~loaded } }
      }
      {
```

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.

```
      \newlist { tabularnotes } { enumerate } { 1 }
      \setlist [ tabularnotes ]
        {
          topsep = 0pt ,
          noitemsep ,
          leftmargin = * ,
          align = left ,
          labelsep = 0pt ,
          label =
            \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
        }
      \newlist { tabularnotes* } { enumerate* } { 1 }
      \setlist [ tabularnotes* ]
        {
          afterlabel = \nobreak ,
          itemjoin = \quad ,
          label =
            \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
        }
```

The command \tabularnote is available in the whole document (and not only in the environments of nicematrix) because we want it to be available in the caption of a {table} (before the following {NiceTabular} or {NiceArray}). That's also the reason why the variables \c@tabularnote and \g_@@_tabularnotes_seq will be cleared at the end of the environment of nicematrix (and not at the beginning).
Unfortunately, if the package caption is loaded, the command \caption evaluates its argument twice and since it is not aware (of course) of \tabularnote, the command \tabularnote is, in fact, not usable in \caption when caption is loaded.[61]

```
      \NewDocumentCommand \tabularnote { m }
        {
          \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
            { \@@_error:n { tabularnote~forbidden } }
            {
```

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

```
              \int_zero:N \l_tmpa_int
              \bool_if:NT \l_@@_notes_detect_duplicates_bool
                {
                  \seq_map_indexed_inline:Nn \g_@@_tabularnotes_seq
                    {
                      \tl_if_eq:nnT { #1 } { ##2 }
                        { \int_set:Nn \l_tmpa_int { ##1 } \seq_map_break: }
                    }
                }
              \int_compare:nNnTF \l_tmpa_int = 0
                {
                  \stepcounter { tabularnote }
                  \seq_put_right:Nx \l_@@_notes_labels_seq
                    { \@@_notes_format:n { \int_use:c { c @ tabularnote } } }
                  \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
                }
                {
                  \seq_put_right:Nx \l_@@_notes_labels_seq
```

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

```
                      { \@@_notes_format:n { \int_use:N \l_tmpa_int } }
                  }
                \peek_meaning:NF \tabularnote
                  {
```

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.

```
                    \hbox_set:Nn \l_tmpa_box
                      {
```

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

```
                        \@@_notes_label_in_tabular:n
                          {
                            \seq_use:Nnnn
                              \l_@@_notes_labels_seq { , } { , } { , }
                          }
                      }
```

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.

```
                    \addtocounter { tabularnote } { -1 }
                    \refstepcounter { tabularnote }
                    \seq_clear:N \l_@@_notes_labels_seq
                    \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).

```
                    \skip_horizontal:n { \box_wd:N \l_tmpa_box }
                  }
              }
          }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
  {
    \begin { pgfscope }
    \pgfset
      {
        outer~sep = \c_zero_dim ,
        inner~sep = \c_zero_dim ,
        minimum~size = \c_zero_dim
      }
    \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
    \pgfnode
      { rectangle }
      { center }
      {
        \vbox_to_ht:nn
          { \dim_abs:n { #5 - #3 } }
          {
            \vfill
            \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
          }
      }
      { #1 }
      { }
    \end { pgfscope }
  }
```

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.

```
\cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
  {
    \begin { pgfscope }
    \pgfset
      {
        outer~sep = \c_zero_dim ,
        inner~sep = \c_zero_dim ,
        minimum~size = \c_zero_dim
      }
    \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
    \pgfpointdiff { #3 } { #2 }
    \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
    \pgfnode
      { rectangle }
      { center }
      {
        \vbox_to_ht:nn
          { \dim_abs:n \l_tmpb_dim }
          { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
      }
      { #1 }
      { }
    \end { pgfscope }
  }
```

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.

```
\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`.

```
\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).

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

```
\dim_new:N \l_@@_inter_dots_dim
\hook_gput_code:nnn { begindocument } { . }
  { \dim_set:Nn \l_@@_inter_dots_dim { 0.45 em } }
```

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

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

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

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

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

```
\dim_new:N \l_@@_radius_dim
\hook_gput_code:nnn { begindocument } { . }
  { \dim_set:Nn \l_@@_radius_dim { 0.53 pt } }
```

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

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

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

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

```
\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).

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

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

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

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

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

```
\dim_new:N \l_@@_notes_above_space_dim
\hook_gput_code:nnn { begindocument } { . }
  { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

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

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

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

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

```
\bool_new:N \l_@@_respect_arraystretch_bool
```

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

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

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

```
\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.

```
\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”.

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

```
\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).

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

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

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

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

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

```
\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 acheive 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.

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

```
\keys_define:nn { NiceMatrix / xdots }
  {
    line-style .code:n =
      {
        \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.

```
          { \cs_if_exist_p:N \tikzpicture }
          { \str_if_eq_p:nn { #1 } { standard } }
          { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
          { \@@_error:n { bad~option~for~line-style } }
      } ,
    line-style .value_required:n = true ,
    color .tl_set:N = \l_@@_xdots_color_tl ,
    color .value_required:n = true ,
    shorten .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        { \dim_set:Nn \l_@@_xdots_shorten_dim { #1 } } ,
```

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

```
    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 `_`.

```
    down .tl_set:N = \l_@@_xdots_down_tl ,
    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 catched when `\Ddots` or `\Iddots` is used (during the construction of the array and not when we draw the dotted lines).

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

```
\keys_define:nn { NiceMatrix / rules }
  {
    color .tl_set:N = \l_@@_rules_color_tl ,
    color .value_required:n = true ,
    width .dim_set:N = \arrayrulewidth ,
    width .value_required:n = true
  }
```

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

```
\keys_define:nn { NiceMatrix / Global }
  {
    custom-line .code:n = \@@_custom_line:n { #1 } ,
    delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
    delimiters .value_required:n = true ,
    rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
    rules .value_required:n = true ,
    standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
    standard-cline .default:n = true ,
    cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
    cell-space-top-limit .value_required:n = true ,
    cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
    cell-space-bottom-limit .value_required:n = true ,
    cell-space-limits .meta:n =
      {
        cell-space-top-limit = #1 ,
        cell-space-bottom-limit = #1 ,
      } ,
    cell-space-limits .value_required:n = true ,
    xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
    light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
    light-syntax .default:n = true ,
    end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
    end-of-row .value_required:n = true ,
    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
    last-row .int_set:N = \l_@@_last_row_int ,
    last-row .default:n = -1 ,
    code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
    code-for-first-col .value_required:n = true ,
    code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
    code-for-last-col .value_required:n = true ,
    code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
    code-for-first-row .value_required:n = true ,
    code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
    code-for-last-row .value_required:n = true ,
    hlines .clist_set:N = \l_@@_hlines_clist ,
    vlines .clist_set:N = \l_@@_vlines_clist ,
    hlines .default:n = all ,
    vlines .default:n = all ,
    vlines-in-sub-matrix .code:n =
      {
        \tl_if_single_token:nTF { #1 }
          { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
          { \@@_error:n { One~letter~allowed } }
      } ,
    vlines-in-sub-matrix .value_required:n = true ,
    hvlines .code:n =
      {
        \clist_set:Nn \l_@@_vlines_clist { all }
        \clist_set:Nn \l_@@_hlines_clist { all }
      } ,
    hvlines-except-borders .code:n =
      {
        \clist_set:Nn \l_@@_vlines_clist { all }
        \clist_set:Nn \l_@@_hlines_clist { all }
        \bool_set_true:N \l_@@_except_borders_bool
      } ,
    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.

```
    renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
    renew-dots .value_forbidden:n = true ,
    nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
    create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
    create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
    create-extra-nodes .meta:n =
      { create-medium-nodes , create-large-nodes } ,
    left-margin .dim_set:N = \l_@@_left_margin_dim ,
    left-margin .default:n = \arraycolsep ,
    right-margin .dim_set:N = \l_@@_right_margin_dim ,
    right-margin .default:n = \arraycolsep ,
    margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
    margin .default:n = \arraycolsep ,
    extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
    extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
    extra-margin .meta:n =
      { extra-left-margin = #1 , extra-right-margin = #1 } ,
    extra-margin .value_required:n = true ,
    respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
    respect-arraystretch .default:n = true
  }
```

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

```
\keys_define:nn { NiceMatrix / Env }
  {
```

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

```
    hvlines-except-corners .code:n =
      {
        \@@_error:n { hvlines-except-corners }
        \group_begin:
        \globaldefs = 1
        \@@_msg_redirect_name:nn { hvlines-except-corners } { none }
        \group_end:
        \clist_set:Nn \l_@@_corners_clist { #1 }
        \clist_set:Nn \l_@@_vlines_clist { all }
        \clist_set:Nn \l_@@_hlines_clist { all }
      } ,
    hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
    corners .clist_set:N = \l_@@_corners_clist ,
    corners .default:n = { NW , SW , NE , SE } ,
    code-before .code:n =
     {
       \tl_if_empty:nF { #1 }
         {
           \tl_put_right:Nn \l_@@_code_before_tl { #1 }
           \bool_set_true:N \l_@@_code_before_bool
         }
     } ,
```

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

```
    c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
    t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
    b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
    baseline .tl_set:N = \l_@@_baseline_tl ,
    baseline .value_required:n = true ,
    columns-width .code:n =
      \tl_if_eq:nnTF { #1 } { auto }
        { \bool_set_true:N \l_@@_auto_columns_width_bool }
        { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
    columns-width .value_required:n = true ,
    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.

```
      \legacy_if:nF { measuring@ }
        {
          \str_set:Nn \l_tmpa_str { #1 }
          \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
```

```
            { \@@_error:nn { Duplicate~name } { #1 } }
            { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
          \str_set_eq:NN \l_@@_name_str \l_tmpa_str
        } ,
    name .value_required:n = true ,
    code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
    code-after .value_required:n = true ,
    colortbl-like .code:n =
      \bool_set_true:N \l_@@_colortbl_like_bool
      \bool_set_true:N \l_@@_code_before_bool ,
    colortbl-like .value_forbidden:n = true
  }
\keys_define:nn { NiceMatrix / notes }
  {
    para .bool_set:N = \l_@@_notes_para_bool ,
    para .default:n = true ,
    code-before .tl_set:N = \l_@@_notes_code_before_tl ,
    code-before .value_required:n = true ,
    code-after .tl_set:N = \l_@@_notes_code_after_tl ,
    code-after .value_required:n = true ,
    bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
    bottomrule .default:n = true ,
    style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
    style .value_required:n = true ,
    label-in-tabular .code:n =
      \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
    label-in-tabular .value_required:n = true ,
    label-in-list .code:n =
      \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
    label-in-list .value_required:n = true ,
    enumitem-keys .code:n =
      {
        \hook_gput_code:nnn { begindocument } { . }
          {
            \bool_if:NT \c_@@_enumitem_loaded_bool
              { \setlist* [ tabularnotes ] { #1 } }
          }
      } ,
    enumitem-keys .value_required:n = true ,
    enumitem-keys-para .code:n =
      {
        \hook_gput_code:nnn { begindocument } { . }
          {
            \bool_if:NT \c_@@_enumitem_loaded_bool
              { \setlist* [ tabularnotes* ] { #1 } }
          }
      } ,
    enumitem-keys-para .value_required:n = true ,
    detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
    detect-duplicates .default:n = true ,
    unknown .code:n  = \@@_error:n { Unknown~key~for~notes }
  }
\keys_define:nn { NiceMatrix / delimiters }
  {
    max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
    max-width .default:n = true ,
    color .tl_set:N = \l_@@_delimiters_color_tl ,
    color .value_required:n = true ,
  }
```

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

```
\keys_define:nn { NiceMatrix }
  {
    NiceMatrixOptions .inherit:n =
      { NiceMatrix / Global } ,
    NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
    NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
    NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
    NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
    SubMatrix / rules .inherit:n = NiceMatrix / rules ,
    CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrix .inherit:n =
      {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
      } ,
    NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
    NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
    NiceTabular .inherit:n =
      {
        NiceMatrix / Global ,
        NiceMatrix / Env
      } ,
    NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
    NiceTabular / rules .inherit:n = NiceMatrix / rules ,
    NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
    NiceArray .inherit:n =
      {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
      } ,
    NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
    NiceArray / rules .inherit:n = NiceMatrix / rules ,
    NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
    pNiceArray .inherit:n =
      {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
      } ,
    pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
    pNiceArray / rules .inherit:n = NiceMatrix / rules ,
    pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
  }
```

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

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

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

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

```
    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.

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

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

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

```
    letter-for-dotted-lines .code:n =
      {
        \@@_error:n { letter-for-dotted-lines }
        \group_begin:
        \globaldefs = 1
        \@@_msg_redirect_name:nn { letter-for-dotted-lines } { none }
        \group_end:
        \tl_if_single_token:nTF { #1 }
          { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
          { \@@_error:n { One~letter~allowed } }
      } ,
    letter-for-dotted-lines .value_required:n = true ,
    notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
    notes .value_required:n = true ,
    sub-matrix .code:n =
      \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
    sub-matrix .value_required:n = true ,
    unknown .code:n  = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
  }
```

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

```
\str_new:N \l_@@_letter_for_dotted_lines_str
```

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

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

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

```
\keys_define:nn { NiceMatrix / NiceMatrix }
  {
    last-col .code:n = \tl_if_empty:nTF {#1}
                         {
                           \bool_set_true:N \l_@@_last_col_without_value_bool
                           \int_set:Nn \l_@@_last_col_int { -1 }
                         }
                         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
    l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
    r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
  }
```

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

```
\keys_define:nn { NiceMatrix / NiceArray }
  {
```

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.

```
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    last-col .code:n = \tl_if_empty:nF { #1 }
                         { \@@_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l_@@_last_col_int ,
    notes / para .bool_set:N = \l_@@_notes_para_bool ,
    notes / para .default:n = true ,
    notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
    notes / bottomrule .default:n = true ,
    tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
    tabularnote .value_required:n = true ,
    r .code:n = \@@_error:n { r~or~l~with~preamble } ,
    l .code:n = \@@_error:n { r~or~l~with~preamble } ,
    unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
  }
\keys_define:nn { NiceMatrix / pNiceArray }
  {
    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
    last-col .code:n = \tl_if_empty:nF {#1}
                         { \@@_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l_@@_last_col_int ,
    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    r .code:n = \@@_error:n { r~or~l~with~preamble } ,
    l .code:n = \@@_error:n { r~or~l~with~preamble } ,
    unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
  }
```

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

```
\keys_define:nn { NiceMatrix / NiceTabular }
  {
```

The dimension `width` will be used if at least a column of type `X` is used. If there is no column of type `X`, an error will be raised.

```
    width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
                    \bool_set_true:N \l_@@_width_used_bool ,
    width .value_required:n = true ,
    notes / para .bool_set:N = \l_@@_notes_para_bool ,
    notes / para .default:n = true ,
    notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
    notes / bottomrule .default:n = true ,
    tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
    tabularnote .value_required:n = true ,
    last-col .code:n = \tl_if_empty:nF {#1}
                         { \@@_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l_@@_last_col_int ,
    r .code:n = \@@_error:n { r~or~l~with~preamble } ,
    l .code:n = \@@_error:n { r~or~l~with~preamble } ,
    unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
  }
```

Important code used by {NiceArrayWithDelims}

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

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

The token list `\g_@@_post_action_cell_tl` will be set during the composition of the box `\l_@@_cell_box` and will be used *after* the composition in order to modify that box (that's why it's called a *post-action*).

```
    \tl_gclear:N \g_@@_post_action_cell_tl
```

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

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

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

```
    \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.

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

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the `\@@_cell_end:` (and the potential `\c_math_toggle_token` also).

```
    \hbox_set:Nw \l_@@_cell_box
    \bool_if:NF \l_@@_NiceTabular_bool
      {
        \c_math_toggle_token
        \bool_if:NT \l_@@_small_bool \scriptstyle
      }
```

For unexplained reason, with XeTeX (and not with the other engines), the environments of nicematrix were all composed in black and do not take into account the color of the encompassing text. As a workaround, you peek the color in force at the beginning of the environment and we use it now (in each cell of the array).

```
    \color { nicematrix }
```

```
    \g_@@_row_style_tl
```

We will call *corners* of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn't always exist simultaneously).
The codes `\l_@@_code_for_first_row_tl` and *al* don't apply in the corners of the matrix.

```
    \int_compare:nNnTF \c@iRow = 0
      {
        \int_compare:nNnT \c@jCol > 0
          {
            \l_@@_code_for_first_row_tl
            \xglobal \colorlet { nicematrix-first-row } { . }
          }
      }
      {
        \int_compare:nNnT \c@iRow = \l_@@_last_row_int
          {
            \l_@@_code_for_last_row_tl
            \xglobal \colorlet { nicematrix-last-row } { . }
          }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_begin_of_row:
  {
    \int_gincr:N \c@iRow
    \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
    \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
    \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgfcoordinate
      { \@@_env: - row - \int_use:N \c@iRow - base }
      { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
    \str_if_empty:NF \l_@@_name_str
      {
        \pgfnodealias
          { \l_@@_name_str - row - \int_use:N \c@iRow - base }
          { \@@_env: - row - \int_use:N \c@iRow - base }
      }
    \endpgfpicture
  }
```

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.

```
\cs_new_protected:Npn \@@_update_for_first_and_last_row:
  {
    \int_compare:nNnTF \c@iRow = 0
      {
        \dim_gset:Nn \g_@@_dp_row_zero_dim
          { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
        \dim_gset:Nn \g_@@_ht_row_zero_dim
          { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
      }
      {
        \int_compare:nNnT \c@iRow = 1
          {
            \dim_gset:Nn \g_@@_ht_row_one_dim
              { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
          }
      }
  }
\cs_new_protected:Npn \@@_rotate_cell_box:
  {
    \box_rotate:Nn \l_@@_cell_box { 90 }
    \int_compare:nNnT \c@iRow = \l_@@_last_row_int
      {
        \vbox_set_top:Nn \l_@@_cell_box
          {
            \vbox_to_zero:n { }
            \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
            \box_use:N \l_@@_cell_box
          }
      }
    \bool_gset_false:N \g_@@_rotate_bool
  }
\cs_new_protected:Npn \@@_adjust_size_box:
  {
    \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
```

```
      {
        \box_set_wd:Nn \l_@@_cell_box
          { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
        \dim_gzero:N \g_@@_blocks_wd_dim
      }
    \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
      {
        \box_set_dp:Nn \l_@@_cell_box
          { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
        \dim_gzero:N \g_@@_blocks_dp_dim
      }
    \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
      {
        \box_set_ht:Nn \l_@@_cell_box
          { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
        \dim_gzero:N \g_@@_blocks_ht_dim
      }
  }
```

```
\cs_new_protected:Npn \@@_cell_end:
  {
    \@@_math_toggle_token:
    \hbox_set_end:
```

The token list `\g_@@_post_action_cell_tl` is (potentially) set during the composition of the box `\l_@@_cell_box` and is used now *after* the composition in order to modify that box.

```
    \g_@@_post_action_cell_tl
    \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
    \@@_adjust_size_box:

    \box_set_ht:Nn \l_@@_cell_box
      { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
    \box_set_dp:Nn \l_@@_cell_box
      { \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").

```
    \dim_gset:Nn \g_@@_max_cell_width_dim
      { \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".

```
    \@@_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.

```
    \bool_if:NTF \g_@@_empty_cell_bool
      { \box_use_drop:N \l_@@_cell_box }
      {
        \bool_lazy_or:nnTF
          \g_@@_not_empty_cell_bool
          { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
          \@@_node_for_cell:
          { \box_use_drop:N \l_@@_cell_box }
      }
    \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
    \bool_gset_false:N \g_@@_empty_cell_bool
    \bool_gset_false:N \g_@@_not_empty_cell_bool
  }
```

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

```
\cs_new_protected:Npn \@@_node_for_cell:
  {
    \pgfpicture
    \pgfsetbaseline \c_zero_dim
    \pgfrememberpicturepositiononpagetrue
    \pgfset
      {
        inner~sep = \c_zero_dim ,
        minimum~width = \c_zero_dim
      }
    \pgfnode
      { rectangle }
      { base }
      { \box_use_drop:N \l_@@_cell_box }
      { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
      { }
    \str_if_empty:NF \l_@@_name_str
      {
        \pgfnodealias
          { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
          { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
      }
    \endpgfpicture
  }
```

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.

```
\cs_new_protected:Npn \@@_patch_node_for_cell:n #1
  {
    \cs_new_protected:Npn \@@_patch_node_for_cell:
      {
        \hbox_set:Nn \l_@@_cell_box
          {
            \box_move_up:nn { \box_ht:N \l_@@_cell_box}
            \hbox_overlap_left:n
              {
                \pgfsys@markposition
                  { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
```

I don't know why the following adjustement 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.

```
                #1
              }
            \box_use:N \l_@@_cell_box
            \box_move_down:nn { \box_dp:N \l_@@_cell_box }
            \hbox_overlap_left:n
```

```
              {
                \pgfsys@markposition
                  { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
                #1
              }
          }
      }
  }
```

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

```
\bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
  {
    \@@_patch_node_for_cell:n
      { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
  }
  { \@@_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}
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & \cdots & \cdots & 6 \\ 7 & \cdots & \cdots & \cdots \end{pmatrix}$$

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.

```
\cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
  {
    \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
      { g_@@_ #2 _ lines _ tl }
      {
        \use:c { @@ _ draw _ #2 : nnn }
          { \int_use:N \c@iRow }
          { \int_use:N \c@jCol }
          { \exp_not:n { #3 } }
      }
  }
\cs_new_protected:Npn \@@_array:
  {
    \bool_if:NTF \l_@@_NiceTabular_bool
      { \dim_set_eq:NN \col@sep \tabcolsep }
      { \dim_set_eq:NN \col@sep \arraycolsep }
    \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
      { \cs_set_nopar:Npn \@halignto { } }
      { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
    \@tabarray
```

`\l_@@_baseline_tl` may have the value `t`, `c` or `b`. However, if the value is `b`, we compose the `\array` (of array) with the option `t` and the right translation will be done further. Remark that `\str_if_eq:VnTF` is fully expandable and you need something fully expandable here.

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

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.

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

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

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

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

```
    \hbox
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \vtop
              {
                \skip_vertical:N 0.5\arrayrulewidth
                \pgfsys@markposition
                  { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
                \skip_vertical:N -0.5\arrayrulewidth
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
          { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
        \str_if_empty:NF \l_@@_name_str
          {
            \pgfnodealias
              { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
              { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
          }
        \endpgfpicture
      }
  }
```

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

```
\cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
```

```
\cs_new_protected:Npn \@@_everycr_i:
  {
    \int_gzero:N \c@jCol
    \bool_gset_false:N \g_@@_after_col_zero_bool
    \bool_if:NF \g_@@_row_of_col_done_bool
      {
        \@@_create_row_node:
```

We don't draw now the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for theses rules (the rules will be drawn by PGF).

```
        \tl_if_empty:NF \l_@@_hlines_clist
          {
            \tl_if_eq:NnF \l_@@_hlines_clist { all }
              {
                \exp_args:NNx
                  \clist_if_in:NnT
                  \l_@@_hlines_clist
                  { \int_eval:n { \c@iRow + 1 } }
              }
              {
```

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.

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

```
                     { \hrule height \arrayrulewidth width \c_zero_dim }
                   }
               }
           }
       }
   }
```

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).

```
\cs_set_protected:Npn \@@_newcolumntype #1
  {
    \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
    \peek_meaning:NTF [
      { \newcol@ #1 }
      { \newcol@ #1 [ 0 ] }
  }
```

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

```
\cs_set_protected:Npn \@@_renew_dots:
  {
    \cs_set_eq:NN \ldots \@@_Ldots
    \cs_set_eq:NN \cdots \@@_Cdots
    \cs_set_eq:NN \vdots \@@_Vdots
    \cs_set_eq:NN \ddots \@@_Ddots
    \cs_set_eq:NN \iddots \@@_Iddots
    \cs_set_eq:NN \dots \@@_Ldots
    \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
  }
```

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

```
\cs_new_protected:Npn \@@_colortbl_like:
  {
    \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
    \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
    \cs_set_eq:NN \columncolor \@@_columncolor_preamble
  }
```

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

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

For unexplained reason, with XeTeX (and not with the other engines), the environments of nicematrix were all composed in black and do not take into account the color of the encompassing text. As a workaround, you peek the color in force at the beginning of the environment and we will it in each cell.

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

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

```
    \int_gzero:N \g_@@_total_X_weight_int
```

```
    \@@_expand_clist:N \l_@@_hlines_clist
    \@@_expand_clist:N \l_@@_vlines_clist
```

If booktabs is loaded, we have to patch the macro \@BTnormal which is a macro of booktabs. The macro \@BTnormal draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro \@BTnormal occurs, the row node has yet been inserted by nicematrix *before* the vertical skip (and thus, at a wrong place). That 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 [62].

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

```
    \bool_if:NT \l_@@_small_bool
      {

        \cs_set_nopar:Npn \arraystretch { 0.47 }
        \dim_set:Nn \arraycolsep { 1.45 pt }
      }
```

```
    \bool_if:NT \g_@@_recreate_cell_nodes_bool
      {
        \tl_put_right:Nn \@@_begin_of_row:
          {
            \pgfsys@markposition
              { \@@_env: - row - \int_use:N \c@iRow - base }
          }
      }
```

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.

```
    \cs_set_nopar:Npn \ialign
      {
        \bool_if:NTF \l_@@_colortbl_loaded_bool
          {
            \CT@everycr
              {
                \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
                \@@_everycr:
              }
          }
          { \everycr { \@@_everycr: } }
        \tabskip = \c_zero_skip
```

The box \@arstrutbox is a box constructed in the beginning of the environment {array}. The construction of that box takes into account the current value of \arraystretch[63] and \extrarowheight (of array). That box is inserted (via \@arstrut) in the beginning of each row of the array. That's why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of \@arstrutbox and that's why we do it in the \ialign.

```
        \dim_gzero_new:N \g_@@_dp_row_zero_dim
        \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
        \dim_gzero_new:N \g_@@_ht_row_zero_dim
        \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
```

[62] cf. \nicematrix@redefine@check@rerun

[63] The option small of nicematrix changes (among others) the value of \arraystretch. This is done, of course, before the call of {array}.

```
        \dim_gzero_new:N \g_@@_ht_row_one_dim
        \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
        \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
        \dim_gzero_new:N \g_@@_ht_last_row_dim
        \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
        \dim_gzero_new:N \g_@@_dp_last_row_dim
        \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 programmation, we will have, in the cells of the array, a clean version of `\ialign`.

```
        \cs_set_eq:NN \ialign \@@_old_ialign:
        \halign
      }
```

We keep in memory the old versions or `\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).

```
    \cs_set_eq:NN \@@_old_ldots \ldots
    \cs_set_eq:NN \@@_old_cdots \cdots
    \cs_set_eq:NN \@@_old_vdots \vdots
    \cs_set_eq:NN \@@_old_ddots \ddots
    \cs_set_eq:NN \@@_old_iddots \iddots
    \bool_if:NTF \l_@@_standard_cline_bool
      { \cs_set_eq:NN \cline \@@_standard_cline }
      { \cs_set_eq:NN \cline \@@_cline }
    \cs_set_eq:NN \Ldots \@@_Ldots
    \cs_set_eq:NN \Cdots \@@_Cdots
    \cs_set_eq:NN \Vdots \@@_Vdots
    \cs_set_eq:NN \Ddots \@@_Ddots
    \cs_set_eq:NN \Iddots \@@_Iddots
    \cs_set_eq:NN \Hline \@@_Hline:
    \cs_set_eq:NN \Hspace \@@_Hspace:
    \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
    \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
    \cs_set_eq:NN \Block \@@_Block:
    \cs_set_eq:NN \rotate \@@_rotate:
    \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
    \cs_set_eq:NN \dotfill \@@_old_dotfill:
    \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
    \cs_set_eq:NN \diagbox \@@_diagbox:nn
    \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
    \cs_set_eq:NN \RowStyle \@@_RowStyle:n
    \seq_map_inline:Nn \l_@@_custom_line_commands_seq
      { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
    \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
    \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:
```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of nicematrix, we patch `{tabular}` to go back to the original definition.

```
    \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
    \hook_gput_code:nnn { env / tabular / begin } { . }
      { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
```

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

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

```
    \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).

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

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

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

```
    \tl_gclear_new:N \g_@@_Cdots_lines_tl
    \tl_gclear_new:N \g_@@_Ldots_lines_tl
    \tl_gclear_new:N \g_@@_Vdots_lines_tl
    \tl_gclear_new:N \g_@@_Ddots_lines_tl
    \tl_gclear_new:N \g_@@_Iddots_lines_tl
    \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

    \tl_gclear_new:N \g_nicematrix_code_before_tl
  }
```

This is the end of \@@_pre_array_ii:.

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

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

```
    \int_compare:nNnT \l_@@_last_row_int = { -1 }
      {
        \bool_set_true:N \l_@@_last_row_without_value_bool
        \bool_if:NT \g_@@_aux_found_bool
          { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
      }
    \int_compare:nNnT \l_@@_last_col_int = { -1 }
      {
        \bool_if:NT \g_@@_aux_found_bool
          { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
      }
```

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

```
    \int_compare:nNnT \l_@@_last_row_int > { -2 }
      {
        \tl_put_right:Nn \@@_update_for_first_and_last_row:
          {
            \dim_gset:Nn \g_@@_ht_last_row_dim
              { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
            \dim_gset:Nn \g_@@_dp_last_row_dim
              { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
          }
      }
```

```
    \seq_gclear:N \g_@@_cols_vlism_seq
    \seq_gclear:N \g_@@_submatrix_seq
```

Now the \CodeBefore.

```
    \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.

```
    \seq_gclear:N \g_@@_pos_of_blocks_seq
```

Idem for other sequences written on the aux file.

```
    \seq_gclear_new:N \g_@@_multicolumn_cells_seq
    \seq_gclear_new:N \g_@@_multicolumn_sizes_seq
```

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

```
    \@@_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.

```
    \box_clear_new:N \l_@@_the_array_box
```

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

```
    \@@_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]).

```
    \dim_zero_new:N \l_@@_left_delim_dim
    \dim_zero_new:N \l_@@_right_delim_dim
    \bool_if:NTF \l_@@_NiceArray_bool
      {
        \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
        \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
      }
      {
```

The command \bBigg@ is a command of amsmath.

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

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).

```
    \hbox_set:Nw \l_@@_the_array_box
    \skip_horizontal:N \l_@@_left_margin_dim
    \skip_horizontal:N \l_@@_extra_left_margin_dim
    \c_math_toggle_token
    \bool_if:NTF \l_@@_light_syntax_bool
      { \use:c { @@-light-syntax } }
      { \use:c { @@-normal-syntax } }
  }
```

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

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

```
    \@@_pre_array:
  }
```

The \CodeBefore

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

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

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

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

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

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

First, the recreation of the row nodes.

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

Now, the recreation of the col nodes.

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

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

```
    \@@_create_diag_nodes:
```

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

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

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

```
    \@@_create_blocks_nodes:
    \bool_if:NT \c_@@_tikz_loaded_bool
      {
        \tikzset
          {
            every~picture / .style =
              { overlay , name~prefix = \@@_env: - }
          }
      }
    \cs_set_eq:NN \cellcolor \@@_cellcolor
    \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
    \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
    \cs_set_eq:NN \rowcolor \@@_rowcolor
    \cs_set_eq:NN \rowcolors \@@_rowcolors
    \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
    \cs_set_eq:NN \arraycolor \@@_arraycolor
    \cs_set_eq:NN \columncolor \@@_columncolor
    \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
    \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
  }
```

```
\cs_new_protected:Npn \@@_exec_code_before:
  {
    \seq_gclear_new:N \g_@@_colors_seq
    \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
    \group_begin:
```

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

```
    \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.

```
    \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.

```
    \@@_actually_color:
    \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
    \group_end:
    \bool_if:NT \g_@@_recreate_cell_nodes_bool
      { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
  }
```

```
\keys_define:nn { NiceMatrix / CodeBefore }
  {
    create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
    create-cell-nodes .default:n = true ,
    sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
    sub-matrix .value_required:n = true ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
  }
\NewDocumentCommand \@@_CodeBefore_keys: { O { } }
  {
    \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
    \@@_CodeBefore:w
  }
```

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.

```
\cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
  {
    \bool_if:NT \g_@@_aux_found_bool
      {
        \@@_pre_code_before:
        #1
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_recreate_cell_nodes:
  {
    \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
      {
        \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
        \pgfcoordinate { \@@_env: - row - ##1 - base }
          { \pgfpointdiff \@@_picture_position: \@@_node_position: }
        \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
          {
            \cs_if_exist:cT
              { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
              {
                \pgfsys@getposition
                  { \@@_env: - ##1 - ####1 - NW }
                  \@@_node_position:
                \pgfsys@getposition
                  { \@@_env: - ##1 - ####1 - SE }
                  \@@_node_position_i:
                \@@_pgf_rect_node:nnn
                  { \@@_env: - ##1 - ####1 }
                  { \pgfpointdiff \@@_picture_position: \@@_node_position: }
                  { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
              }
          }
      }
    \int_step_inline:nn \c@iRow
      {
        \pgfnodealias
          { \@@_env: - ##1 - last }
          { \@@_env: - ##1 - \int_use:N \c@jCol }
      }
    \int_step_inline:nn \c@jCol
      {
        \pgfnodealias
          { \@@_env: - last - ##1 }
          { \@@_env: - \int_use:N \c@iRow - ##1 }
      }
    \@@_create_extra_nodes:
  }
```

```
\cs_new_protected:Npn \@@_create_blocks_nodes:
  {
    \pgfpicture
    \pgf@relevantforpicturesizefalse
    \pgfrememberpicturepositiononpagetrue
    \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
      { \@@_create_one_block_node:nnnnn ##1 }
    \endpgfpicture
  }
```

The following command is called `\@@_create_one_block_node:nnnnn` but, in fact, it creates a node only if the last argument (`#5`) which is the name of the block, is not empty.[64]

```
\cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
  {
    \tl_if_empty:nF { #5 }
      {
        \@@_qpoint:n { col - #2 }
        \dim_set_eq:NN \l_tmpa_dim \pgf@x
        \@@_qpoint:n { #1 }
        \dim_set_eq:NN \l_tmpb_dim \pgf@y
        \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
        \@@_qpoint:n { \int_eval:n { #3 + 1 } }
        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
        \@@_pgf_rect_node:nnnnn
          { \@@_env: - #5 }
          { \dim_use:N \l_tmpa_dim }
          { \dim_use:N \l_tmpb_dim }
          { \dim_use:N \l_@@_tmpc_dim }
          { \dim_use:N \l_@@_tmpd_dim }
      }
  }
```

```
\cs_new_protected:Npn \@@_patch_for_revtex:
  {
    \cs_set_eq:NN \@addamp \@addamp@LaTeX
    \cs_set_eq:NN \insert@column \insert@column@array
    \cs_set_eq:NN \@classx \@classx@array
    \cs_set_eq:NN \@xarraycr \@xarraycr@array
    \cs_set_eq:NN \@arraycr \@arraycr@array
    \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
    \cs_set_eq:NN \array \array@array
    \cs_set_eq:NN \@array \@array@array
    \cs_set_eq:NN \@tabular \@tabular@array
    \cs_set_eq:NN \@mkpream \@mkpream@array
    \cs_set_eq:NN \endarray \endarray@array
    \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
    \cs_set:Npn \endtabular { \endarray $\egroup} % $
  }
```

[64] Moreover, there is also in the list `\g_@@_pos_of_blocks_seq` the positions of the dotted lines (created by `\Cdots`, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).

The environment {NiceArrayWithDelims}

```
\NewDocumentEnvironment { NiceArrayWithDelims }
  { m m O { } m ! O { } t \CodeBefore }
  {
    \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:

    \@@_provide_pgfsyspdfmark:
    \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.

```
    \bgroup


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


    \int_gzero:N \g_@@_block_box_int
    \dim_zero:N \g_@@_width_last_col_dim
    \dim_zero:N \g_@@_width_first_col_dim
    \bool_gset_false:N \g_@@_row_of_col_done_bool
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
    \bool_if:NTF \l_@@_NiceTabular_bool
      \mode_leave_vertical:
      \@@_test_if_math_mode:
    \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
    \bool_set_true:N \l_@@_in_env_bool
```

The command `\CT@arc@` contains the instruction of color for the rules of the array[65]. 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.

```
      \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.

```
    \cs_if_exist:NT \tikz@library@external@loaded
      {
        \tikzexternaldisable
        \cs_if_exist:NT \ifstandalone
          { \tikzset { external / optimize = false } }
      }
```

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

```
    \int_gincr:N \g_@@_env_int
    \bool_if:NF \l_@@_block_auto_columns_width_bool
      { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

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

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

```
    \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
    \seq_gclear:N \g_@@_pos_of_xdots_seq
    \tl_gclear_new:N \g_@@_code_before_tl
```

[65] e.g. `\color[rgb]{0.5,0.5,0}`

```
    \tl_gclear:N \g_@@_row_style_tl
```

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

```
    \bool_gset_false:N \g_@@_aux_found_bool
    \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
      {
        \bool_gset_true:N \g_@@_aux_found_bool
        \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
      }
```

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.

```
     \tl_gclear:N \g_@@_aux_tl
    \tl_if_empty:NF \g_@@_code_before_tl
      {
        \bool_set_true:N \l_@@_code_before_bool
        \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
      }
```

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.

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

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

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

```
    \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
  }
  {
    \bool_if:NTF \l_@@_light_syntax_bool
      { \use:c { end @@-light-syntax } }
      { \use:c { end @@-normal-syntax } }
    \c_math_toggle_token
    \skip_horizontal:N \l_@@_right_margin_dim
    \skip_horizontal:N \l_@@_extra_right_margin_dim
    \hbox_set_end:
```

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

If the user has used the key width without any column X, we raise an error.

```
    \bool_if:NT \l_@@_width_used_bool
      {
        \int_compare:nNnT \g_@@_total_X_weight_int = 0
          { \@@_error:n { width~without~X~columns } }
      }
```

Now, if there is at least one X-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, l_@@_X_columns_dim will be the width of a column of weight 1. For a X-column of weight n, the width will be l_@@_X_columns_dim multiplied by n.

```
    \int_compare:nNnT \g_@@_total_X_weight_int > 0
      {
```

```
    \tl_gput_right:Nx \g_@@_aux_tl
      {
        \bool_set_true:N \l_@@_X_columns_aux_bool
        \dim_set:Nn \l_@@_X_columns_dim
          {
            \dim_compare:nNnTF
              {
                \dim_abs:n
                  { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
              }
              <
              { 0.001 pt }
              { \dim_use:N \l_@@_X_columns_dim }
              {
                \dim_eval:n
                  {
                    ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
                    / \int_use:N \g_@@_total_X_weight_int
                    + \l_@@_X_columns_dim
                  }
              }
          }
      }
  }
```

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).

```
\int_compare:nNnT \l_@@_last_row_int > { -2 }
  {
    \bool_if:NF \l_@@_last_row_without_value_bool
      {
        \int_compare:nNnF \l_@@_last_row_int = \c@iRow
          {
            \@@_error:n { Wrong~last~row }
            \int_gset_eq:NN \l_@@_last_row_int \c@iRow
          }
      }
  }
```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` change: `\c@jCol` will be the number of columns without the "last column"; `\g_@@_col_total_int` will be the number of columns with this "last column".[66]

```
\int_gset_eq:NN \c@jCol \g_@@_col_total_int
\bool_if:nTF \g_@@_last_col_found_bool
  { \int_gdecr:N \c@jCol }
  {
    \int_compare:nNnT \l_@@_last_col_int > { -1 }
      { \@@_error:n { last~col~not~used } }
  }
```

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

```
\int_gset_eq:NN \g_@@_row_total_int \c@iRow
\int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }
```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential "first column" (we remind that this "first column" has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 129).

```
\int_compare:nNnT \l_@@_first_col_int = 0
  {
    \skip_horizontal:N \col@sep
    \skip_horizontal:N \g_@@_width_first_col_dim
```

[66] We remind that the potential "first column" (exterior) has the number 0.

```
 }
```

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.

```
\bool_if:NTF \l_@@_NiceArray_bool
 {
  \str_case:VnF \l_@@_baseline_tl
   {
    b \@@_use_arraybox_with_notes_b:
    c \@@_use_arraybox_with_notes_c:
   }
   \@@_use_arraybox_with_notes:
 }
```

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).

```
 {
  \int_compare:nNnTF \l_@@_first_row_int = 0
   {
    \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
    \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
   }
   { \dim_zero:N \l_tmpa_dim }
```

We compute `\l_tmpb_dim` which is the total height of the “last row” below the array (when the key `last-row` is used). A value of -2 for `\l_@@_last_row_int` means that there is no “last row”.[67]

```
  \int_compare:nNnTF \l_@@_last_row_int > { -2 }
   {
    \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
    \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
   }
   { \dim_zero:N \l_tmpb_dim }
  \hbox_set:Nn \l_tmpa_box
   {
    \c_math_toggle_token
    \tl_if_empty:NF \l_@@_delimiters_color_tl
     { \color { \l_@@_delimiters_color_tl } }
    \exp_after:wN \left \g_@@_left_delim_tl
    \vcenter
     {
```

We take into account the “first row” (we have previously computed its total height in `\l_tmpa_dim`). The `\hbox:n` (or `\hbox`) is necessary here. There was a bug in the following line (corrected the 2021/11/23).

```
      \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
      \hbox
       {
        \bool_if:NTF \l_@@_NiceTabular_bool
         { \skip_horizontal:N -\tabcolsep }
         { \skip_horizontal:N -\arraycolsep }
        \@@_use_arraybox_with_notes_c:
        \bool_if:NTF \l_@@_NiceTabular_bool
         { \skip_horizontal:N -\tabcolsep }
         { \skip_horizontal:N -\arraycolsep }
       }
```

We take into account the “last row” (we have previously computed its total height in `\l_tmpb_dim`). There was a bug in the following line (corrected the 2021/11/23).

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

[67] A value of -1 for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

Curiously, we have to put again the following specification of color. Otherwise, with XeLaTeX (and not with the other engines), the closing delimiter is not colored.

```
            \tl_if_empty:NF \l_@@_delimiters_color_tl
              { \color { \l_@@_delimiters_color_tl } }
            \exp_after:wN \right \g_@@_right_delim_tl
            \c_math_toggle_token
          }
```

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.

```
        \bool_if:NTF \l_@@_delimiters_max_width_bool
          {
            \@@_put_box_in_flow_bis:nn
              \g_@@_left_delim_tl \g_@@_right_delim_tl
          }
          \@@_put_box_in_flow:
      }
```

We take into account a potential "last column" (this "last column" has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 130).

```
    \bool_if:NT \g_@@_last_col_found_bool
      {
        \skip_horizontal:N \g_@@_width_last_col_dim
        \skip_horizontal:N \col@sep
      }
    \bool_if:NF \l_@@_Matrix_bool
      {
        \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
          { \@@_error:n { columns~not~used } }
      }
    \group_begin:
    \globaldefs = 1
    \@@_msg_redirect_name:nn { columns~not~used } { error }
    \group_end:
    \@@_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.

```
    \egroup
```

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

```
    \iow_now:Nn \@mainaux { \ExplSyntaxOn }
    \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 }  }
    \iow_now:Nx \@mainaux
      {
        \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
          { \exp_not:V \g_@@_aux_tl }
      }
    \iow_now:Nn \@mainaux { \ExplSyntaxOff }


    \bool_if:NT \c_@@_footnote_bool \endsavenotes
  }
```

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.

```
\cs_new_protected:Npn \@@_construct_preamble:
  {
```

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 the L3 programming layer.

We redefine the column types w and W. We use \@@_newcolumntype instead of \newcolumtype because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters w and W. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able to use the standard column types w and W in potential {tabular} of array in some cells of our array. That’s why we do those redefinitions in a TeX group.

```
    \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).

```
    \bool_if:NF \l_@@_Matrix_bool
     {
       \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
       \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
```

If the package varwidth has defined the column type V, we protect from expansion by redefining it to \@@_V: (which will be catched by our system).

```
      \cs_if_exist:NT \NC@find@V { \@@_newcolumntype V { \@@_V: } }
```

First, we have to store our preamble in the token register \@temptokena (those “token registers” are *not* supported by the L3 programming layer).

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

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

```
      \@tempswatrue
```

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

```
      \@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.

```
      \int_gzero:N \c@jCol
      \tl_gclear:N \g_@@_preamble_tl
```

\g_tmpb_bool will be raised if you have a | at the end of the preamble.

```
      \bool_gset_false:N \g_tmpb_bool
      \tl_if_eq:NnTF \l_@@_vlines_clist { all }
        {
          \tl_gset:Nn \g_@@_preamble_tl
            { ! { \skip_horizontal:N \arrayrulewidth } }
        }
        {
          \clist_if_in:NnT \l_@@_vlines_clist 1
            {
              \tl_gset:Nn \g_@@_preamble_tl
                { ! { \skip_horizontal:N \arrayrulewidth } }
            }
        }
```

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

```
        \seq_clear:N \g_@@_cols_vlism_seq
```

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

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

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

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

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

```
    \bool_if:NT \l_@@_colortbl_like_bool
      {
        \regex_replace_all:NnN
          \c_@@_columncolor_regex
          { \c { @@_columncolor_preamble } }
          \g_@@_preamble_tl
      }
```

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

```
    \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}`.

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

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

```
    \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }
```

We complete the preamble with the potential "exterior columns" (on both sides).

```
    \int_compare:nNnTF \l_@@_first_col_int = 0
      { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
      {
        \bool_lazy_all:nT
          {
            \l_@@_NiceArray_bool
            { \bool_not_p:n \l_@@_NiceTabular_bool }
            { \tl_if_empty_p:N \l_@@_vlines_clist }
            { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
          }
          { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
      }
    \int_compare:nNnTF \l_@@_last_col_int > { -1 }
      { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
      {
        \bool_lazy_all:nT
          {
            \l_@@_NiceArray_bool
            { \bool_not_p:n \l_@@_NiceTabular_bool }
            { \tl_if_empty_p:N \l_@@_vlines_clist }
            { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
          }
          { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
      }
```

We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it's not possible to do that in `{NiceTabular*}` (`\l_@@_tabular_width_dim`=0pt).

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

```
          { > { \@@_error_too_much_cols: } l }
      }
  }
```

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

```
\cs_new_protected:Npn \@@_patch_preamble:n #1
  {
    \str_case:nnF { #1 }
      {
        c { \@@_patch_preamble_i:n #1 }
        l { \@@_patch_preamble_i:n #1 }
        r { \@@_patch_preamble_i:n #1 }
        > { \@@_patch_preamble_ii:nn #1 }
        ! { \@@_patch_preamble_ii:nn #1 }
        @ { \@@_patch_preamble_ii:nn #1 }
        | { \@@_patch_preamble_iii:n #1 }
        p { \@@_patch_preamble_iv:n #1 }
        b { \@@_patch_preamble_iv:n #1 }
        m { \@@_patch_preamble_iv:n #1 }
        \@@_V:  { \@@_patch_preamble_v:n }
        V { \@@_patch_preamble_v:n }
        \@@_w: { \@@_patch_preamble_vi:nnnn { }                          #1 }
        \@@_W: { \@@_patch_preamble_vi:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
        \@@_S: { \@@_patch_preamble_vii:n }
        (  { \@@_patch_preamble_viii:nn #1 }
        [  { \@@_patch_preamble_viii:nn #1 }
        \{ { \@@_patch_preamble_viii:nn #1 }
        )  { \@@_patch_preamble_ix:nn #1 }
        ]  { \@@_patch_preamble_ix:nn #1 }
        \} { \@@_patch_preamble_ix:nn #1 }
        X  { \@@_patch_preamble_x:n }
```

When tabularx is loaded, a local redefinition of the specifier `X` is done to replace `X` by `\@@_X`. Thus, our column type `X` will be used in the `{NiceTabularX}`.

```
        \@@_X { \@@_patch_preamble_x:n }
        \q_stop { }
      }
      {
        \str_case_e:nnF { #1 }
          {
            \l_@@_letter_for_dotted_lines_str
              { \@@_patch_preamble_xii:n #1 }
            \l_@@_letter_vlism_tl
              {
                \seq_gput_right:Nx \g_@@_cols_vlism_seq
                  { \int_eval:n { \c@jCol + 1 } }
                \tl_gput_right:Nx \g_@@_preamble_tl
                  { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
                \@@_patch_preamble:n
              }
          }
```

Now the case of a letter set by the final user for a customized rule. Such customized rule is defined by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs. Among the keys avalaible in that list, there is the key `letter`. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix/ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```
          {
            \keys_set_known:nnN { NiceMatrix / ColumnTypes } { #1 } \l_tmpa_tl
            \tl_if_empty:NTF \l_tmpa_tl
```

```
              \@@_patch_preamble:n
              { \@@_fatal:nn { unknown~column~type } { #1 } }
          }
      }
  }
```

Now, we will list all the auxiliary functions for the different types of entries in the preamble of the array.

For c, l and r

```
\cs_new_protected:Npn \@@_patch_preamble_i:n #1
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
        #1
        < \@@_cell_end:
      }
```

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

```
    \int_gincr:N \c@jCol
    \@@_patch_preamble_xi:n
  }
```

For >, ! and @

```
\cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
  {
    \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
    \@@_patch_preamble:n
  }
```

For |

```
\cs_new_protected:Npn \@@_patch_preamble_iii:n #1
  {
```

\l_tmpa_int is the number of successive occurrences of |

```
    \int_incr:N \l_tmpa_int
    \@@_patch_preamble_iii_i:n
  }
\cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
  {
    \str_if_eq:nnTF { #1 } |
      { \@@_patch_preamble_iii:n | }
      {
        \tl_gput_right:Nx \g_@@_preamble_tl
          {
            \exp_not:N !
              {
                \skip_horizontal:n
                  {
```

Here, the command \dim_eval:n is mandatory.

```
                    \dim_eval:n
                      {
                        \arrayrulewidth * \l_tmpa_int
                        + \doublerulesep * ( \l_tmpa_int - 1)
                      }
                  }
              }
          }
        \tl_gput_right:Nx \g_@@_internal_code_after_tl
          {
            \@@_vline:n
              {
                position = \int_eval:n { \c@jCol + 1 } ,
                multiplicity = \int_use:N \l_tmpa_int ,
              }
```

We don't have provided value for start nor for end, which means that the rule will cover (potentially) all the rows of the array.

```
          }
        \int_zero:N \l_tmpa_int
        \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
        \@@_patch_preamble:n #1
      }
  }
\bool_new:N \l_@@_bar_at_end_of_pream_bool
```

The specifier p (and also the specifiers m and b) have an optional argument between square brackets for a list of *key-value* pairs. Here are the corresponding keys. This set of keys will also be used by the X columns.

```
\keys_define:nn { WithArrows / p-column }
  {
    r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
    r .value_forbidden:n = true ,
    c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
    c .value_forbidden:n = true ,
    l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
    l .value_forbidden:n = true ,
    si .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
    si .value_forbidden:n = true ,
    p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
    p .value_forbidden:n = true ,
    t .meta:n = p ,
    m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
    m .value_forbidden:n = true ,
    b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
    b .value_forbidden:n = true ,
  }
```

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

```
\cs_new_protected:Npn \@@_patch_preamble_iv:n #1
  {
    \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```
    \@@_patch_preamble_iv_i:n
  }
\cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
  {
    \str_if_eq:nnTF { #1 } { [ }
      { \@@_patch_preamble_iv_ii:w [ }
      { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
  }
\cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
  { \@@_patch_preamble_iv_iii:nn { #1 } }
```

#1 is the optional argument of the specifier (a list of *key-value* pairs).
#2 is the mandatory argument of the specifier: the width of the column.

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

The possible values of \l_@@_hpos_col_str are j (for *justified* which is the initial value), l, c and r (when the user has used the corresponding key in the optional argument of the specifier).

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

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

```
\cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
  {
    \use:x
      {
        \@@_patch_preamble_iv_v:nnnnnnnn
          { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
          { \dim_eval:n { #1 } }
          {
```

The parameter \l_@@_hpos_col_str (as \l_@@_vpos_col_str) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter \l_@@_hpos_cell_str which will provide the horizontal alignment of the column to which belongs the cell.

```
            \str_if_eq:VnTF \l_@@_hpos_col_str j
              { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
              {
                \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
                  { \l_@@_hpos_col_str }
              }
            \str_case:Vn \l_@@_hpos_col_str
              {
                c { \exp_not:N \centering }
                l { \exp_not:N \raggedright }
                r { \exp_not:N \raggedleft }
              }
          }
          { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
          { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
          { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
          { #2 }
          {
            \str_case:VnF \l_@@_hpos_col_str
              {
                { j } { c }
                { si } { c }
              }
              { \l_@@_hpos_col_str }
          }
      }
```

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

```
    \int_gincr:N \c@jCol
    \@@_patch_preamble_xi:n
  }
```

#1 is the optional argument of {minipage} (or {varwidth}): t of b. Indeed, for the columns of type m, we use the value b here because there is a special post-action in order to center vertically the box (see #4).
#2 is the width of the {minipage} (or {varwidth}), that is to say also the width of the column.
#3 is the coding for the horizontal position of the content of the cell (\centering, \raggedright, \raggedleft or nothing). It's also possible to put in that #3 some code to fix the value of \l_@@_hpos_cell_str which will be available in each cell of the column.
#4 is an extra-code which contains \@@_center_cell_box: (when the column is a m column) or nothing (in the other cases).

#5 is a code put just before the c (or r or l: see #8).
#6 is a code put just after the c (or r or l: see #8).
#7 is the type of environment: minipage or varwidth.
#8 is the lettre c or r or l which is the basic specificier of column which is used *in fine*.

```
\cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > {
```

The parameter \l_@@_col_width_dim, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

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

The following lines have been taken from array.sty.

```
            \everypar
              {
                \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
                \everypar { }
              }
```

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

```
            #3
```

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

```
            \g_@@_row_style_tl
            \arraybackslash
            #5
          }
        #8
        < {
            #6
```

The following line has been taken from array.sty.

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

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

```
            #4
            \@@_cell_end:
          }
      }
  }
```

The following command will be used in m-columns in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in array) that if the height of the cell is no more that the height of \@arstrutbox, there is only one row.

```
\cs_new_protected:Npn \@@_center_cell_box:
  {
```

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

```
    \tl_gput_right:Nn \g_@@_post_action_cell_tl
      {
        \int_compare:nNnT
          { \box_ht:N \l_@@_cell_box }
          >
          { \box_ht:N \@arstrutbox }
          {
            \hbox_set:Nn \l_@@_cell_box
              {
                \box_move_down:nn
                  {
                    ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
                    + \baselineskip ) / 2
                  }
                  { \box_use:N \l_@@_cell_box }
              }
          }
      }
  }
```

For V (similar to the V of varwidth).

```
\cs_new_protected:Npn \@@_patch_preamble_v:n #1
  {
    \str_if_eq:nnTF { #1 } { [ }
      { \@@_patch_preamble_v_i:w [ }
      { \@@_patch_preamble_v_i:w [ ] { #1 } }
  }
\cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
  { \@@_patch_preamble_v_ii:nn { #1 } }
\cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
  {
    \str_set:Nn \l_@@_vpos_col_str { p }
    \str_set:Nn \l_@@_hpos_col_str { j }
    \keys_set:nn { WithArrows / p-column } { #1 }
    \bool_if:NTF \c_@@_varwidth_loaded_bool
      { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
      {
        \@@_error:n { varwidth~not~loaded }
        \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
      }
  }
```

For w and W

```
\cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > {
```

The parameter \l_@@_col_width_dim, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

```
            \dim_set:Nn \l_@@_col_width_dim { #4 }
            \hbox_set:Nw \l_@@_cell_box
            \@@_cell_begin:w
            \str_set:Nn \l_@@_hpos_cell_str { #3 }
          }
        c
        < {
            \@@_cell_end:
            #1
            \hbox_set_end:
            \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
            \@@_adjust_size_box:
            \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
          }
      }
```

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

```
    \int_gincr:N \c@jCol
    \@@_patch_preamble_xi:n
  }
```

For `\@@_S:`. If the user has used `S[...]`, `S` has been replaced by `\@@_S:` during the first expansion of the preamble (done with the tools of standard LaTeX and array).

```
\cs_new_protected:Npn \@@_patch_preamble_vii:n #1
  {
    \str_if_eq:nnTF { #1 } { [ }
      { \@@_patch_preamble_vii_i:w [ }
      { \@@_patch_preamble_vii_i:w [ ] { #1 } }
  }
\cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
  { \@@_patch_preamble_vii_ii:n { #1 } }
\cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
  {
```

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

```
    \cs_if_exist:NTF \siunitx_cell_begin:w
      {
        \tl_gput_right:Nn \g_@@_preamble_tl
          {
            > {
                \@@_cell_begin:w
                \keys_set:nn { siunitx } { #1 }
                \siunitx_cell_begin:w
              }
            c
            < { \siunitx_cell_end: \@@_cell_end: }
          }
```

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

```
        \int_gincr:N \c@jCol
        \@@_patch_preamble_xi:n
      }
      { \@@_fatal:n { Version~of~siunitx~too~old } }
  }
```

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

```
\cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
  {
    \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.

```
    \int_compare:nNnTF \c@jCol = \c_zero_int
      {
        \str_if_eq:VnTF \g_@@_left_delim_tl { . }
          {
```

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

```
            \tl_gset:Nn \g_@@_left_delim_tl { #1 }
            \tl_gset:Nn \g_@@_right_delim_tl { . }
            \@@_patch_preamble:n #2
          }
          {
            \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
            \@@_patch_preamble_viii_i:nn { #1 } { #2 }
          }
      }
      { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
  }
\cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
  {
    \tl_gput_right:Nx \g_@@_internal_code_after_tl
      { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
    \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
      {
        \@@_error:nn { delimiter~after~opening } { #2 }
        \@@_patch_preamble:n
      }
      { \@@_patch_preamble:n #2 }
  }
```

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}).

```
\cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
  {
    \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
    \tl_if_in:nnTF { ) ] \} } { #2 }
      { \@@_patch_preamble_ix_i:nnn #1 #2 }
      {
        \tl_if_eq:nnTF { \q_stop } { #2 }
          {
            \str_if_eq:VnTF \g_@@_right_delim_tl { . }
              { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
              {
                \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
                \tl_gput_right:Nx \g_@@_internal_code_after_tl
                  { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
                \@@_patch_preamble:n #2
              }
          }
          {
            \tl_if_in:nnT { ( [ \{ } { #2 }
              { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
            \tl_gput_right:Nx \g_@@_internal_code_after_tl
              { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
            \@@_patch_preamble:n #2
          }
      }
  }
\cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
  {
    \tl_if_eq:nnTF { \q_stop } { #3 }
      {
        \str_if_eq:VnTF \g_@@_right_delim_tl { . }
          {
            \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
            \tl_gput_right:Nx \g_@@_internal_code_after_tl
              { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
            \tl_gset:Nn \g_@@_right_delim_tl { #2 }
          }
          {
            \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
            \tl_gput_right:Nx \g_@@_internal_code_after_tl
              { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
            \@@_error:nn { double~closing~delimiter } { #2 }
          }
      }
      {
        \tl_gput_right:Nx \g_@@_internal_code_after_tl
          { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
        \@@_error:nn { double~closing~delimiter } { #2 }
        \@@_patch_preamble:n #3
      }
  }
```

For the case of a letter X. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a [after the letter X.

```
\cs_new_protected:Npn \@@_patch_preamble_x:n #1
  {
    \str_if_eq:nnTF { #1 } { [ }
      { \@@_patch_preamble_x_i:w [ }
      { \@@_patch_preamble_x_i:w [ ] #1 }
  }
\cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
  { \@@_patch_preamble_x_ii:n { #1 } }
```

#1 is the optional argument of the X specifier (a list of *key-value* pairs).

The following set of keys is for the specifier X in the preamble of the array. Such specifier may have as keys all the keys of { WithArrows / p-column } but also a key as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter \l_@@_weight_int).

```
\keys_define:nn { WithArrows / X-column }
  { unknown .code:n = \int_set:Nn \l_@@_weight_int { \l_keys_key_str } }
```

In the following command, #1 is the list of the options of the specifier X.

```
\cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
  {
```

The possible values of \l_@@_hpos_col_str are j (for *justified* which is the initial value), l, c and r (when the user has used the corresponding key in the optional argument of the specifier X).

```
    \str_set:Nn \l_@@_hpos_col_str { j }
```

The possible values of \l_@@_vpos_col_str are p (the initial value), m and b (when the user has used the corresponding key in the optional argument of the specifier X).

```
    \tl_set:Nn \l_@@_vpos_col_str { p }
```

The integer \l_@@_weight_int will be the weight of the X column (the initial value is 1). The user may specify a different value (such as 2, 3, etc.) by putting that value in the optional argument of the specifier. The weights of the X columns are used in the computation of the actual width of those columns as in tabu of tabularray.

```
    \int_zero_new:N \l_@@_weight_int
    \int_set:Nn \l_@@_weight_int { 1 }
    \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
    \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
    \int_compare:nNnT \l_@@_weight_int < 0
      {
        \@@_error:nx { negative~weight } { \int_use:N \l_@@_weight_int }
        \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
      }
    \int_gadd:Nn \g_@@_total_X_weight_int \l_@@_weight_int
```

We test whether we know the width of the X-columns by reading the aux file (after the first compilation, the width of the X-columns is computed and written in the aux file).

```
    \bool_if:NTF \l_@@_X_columns_aux_bool
      {
        \@@_patch_preamble_iv_iv:nn
          { \l_@@_weight_int \l_@@_X_columns_dim }
          { minipage }
      }
      {
        \tl_gput_right:Nn \g_@@_preamble_tl
          {
            > {
                \@@_cell_begin:w
                \bool_set_true:N \l_@@_X_column_bool
```

The following code will nullify the box of the cell.

```
                \tl_gput_right:Nn \g_@@_post_action_cell_tl
                  { \hbox_set:Nn \l_@@_cell_box { } }
```

We put a `{minipage}` to give to the user the ability to put a command such as `\centering` in the `\RowStyle`.

```
                \begin { minipage } { 5 cm } \arraybackslash
              }
            c
            < {
                \end { minipage }
                \@@_cell_end:
              }
          }
        \int_gincr:N \c@jCol
        \@@_patch_preamble_xi:n
      }
  }
```

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

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

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.

```
\cs_new_protected:Npn \@@_patch_preamble_xi:n #1
  {
    \str_if_eq:nnTF { #1 } { < }
      \@@_patch_preamble_xiii:n
      {
        \tl_if_eq:NnTF \l_@@_vlines_clist { all }
          {
            \tl_gput_right:Nn \g_@@_preamble_tl
              { ! { \skip_horizontal:N \arrayrulewidth } }
          }
          {
            \exp_args:NNx
            \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
              {
                \tl_gput_right:Nn \g_@@_preamble_tl
                  { ! { \skip_horizontal:N \arrayrulewidth } }
              }
          }
        \@@_patch_preamble:n { #1 }
      }
  }
```

```
\cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
  {
    \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
    \@@_patch_preamble_xi:n
  }
```

The redefinition of \multicolumn

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

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

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

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

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

```
    \@temptokena = { #2 }
    \@tempswatrue
    \@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.

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

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

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

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

```
    \int_compare:nNnT { #1 } > 1
      {
        \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
          { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
        \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
        \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
          {
            {
              \int_compare:nNnTF \c@jCol = 0
                { \int_eval:n { \c@iRow + 1 } }
                { \int_use:N \c@iRow }
            } % modified 2022/01/10
            { \int_eval:n { \c@jCol + 1 } }
            {
              \int_compare:nNnTF \c@jCol = 0
                { \int_eval:n { \c@iRow + 1 } }
                { \int_use:N \c@iRow }
            } % modified 2022/01/10
            { \int_eval:n { \c@jCol + #1 } }
            { } % for the name of the block
          }
      }
```

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

```
    \cs_set:Npn \@sharp { #3 }
    \@arstrut
    \@preamble
    \null
```

We add some lines.

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

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.

```
\cs_new_protected:Npn \@@_patch_m_preamble:n #1
  {
    \str_case:nnF { #1 }
      {
        c { \@@_patch_m_preamble_i:n #1 }
        l { \@@_patch_m_preamble_i:n #1 }
        r { \@@_patch_m_preamble_i:n #1 }
        > { \@@_patch_m_preamble_ii:nn #1 }
        ! { \@@_patch_m_preamble_ii:nn #1 }
        @ { \@@_patch_m_preamble_ii:nn #1 }
        | { \@@_patch_m_preamble_iii:n #1 }
        p { \@@_patch_m_preamble_iv:nnn t #1 }
        m { \@@_patch_m_preamble_iv:nnn c #1 }
        b { \@@_patch_m_preamble_iv:nnn b #1 }
        \@@_w: { \@@_patch_m_preamble_v:nnnn { }                          #1 }
        \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
        \q_stop { }
      }
      { \@@_fatal:nn { unknown~column~type } { #1 } }
  }
```

For c, l and r

```
\cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
        #1
        < \@@_cell_end:
      }
```

We test for the presence of a <.

```
    \@@_patch_m_preamble_x:n
  }
```

For >, ! and @

```
\cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
  {
    \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
    \@@_patch_m_preamble:n
  }
```

For |

```
\cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
  {
    \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
    \@@_patch_m_preamble:n
  }
```

For p, m and b

```
\cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > {
            \@@_cell_begin:w
            \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
            \mode_leave_vertical:
            \arraybackslash
            \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
          }
        c
        < {
            \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
            \end { minipage }
            \@@_cell_end:
          }
      }
```

We test for the presence of a <.

```
    \@@_patch_m_preamble_x:n
  }
```

For w and W

```
\cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > {
            \hbox_set:Nw \l_@@_cell_box
            \@@_cell_begin:w
            \str_set:Nn \l_@@_hpos_cell_str { #3 }
          }
        c
        < {
            \@@_cell_end:
            #1
            \hbox_set_end:
            \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
            \@@_adjust_size_box:
            \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
          }
      }
```

We test for the presence of a <.

```
    \@@_patch_m_preamble_x:n
  }
```

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.

```
\cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
  {
    \str_if_eq:nnTF { #1 } { < }
      \@@_patch_m_preamble_ix:n
      {
        \tl_if_eq:NnTF \l_@@_vlines_clist { all }
          {
            \tl_gput_right:Nn \g_@@_preamble_tl
              { ! { \skip_horizontal:N \arrayrulewidth } }
          }
          {
            \exp_args:NNx
            \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
              {
                \tl_gput_right:Nn \g_@@_preamble_tl
                  { ! { \skip_horizontal:N \arrayrulewidth } }
              }
          }
        \@@_patch_m_preamble:n { #1 }
      }
  }
\cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
  {
    \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
    \@@_patch_m_preamble_x:n
  }
```

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`).

```
\cs_new_protected:Npn \@@_put_box_in_flow:
  {
    \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
    \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
    \tl_if_eq:NnTF \l_@@_baseline_tl { c }
      { \box_use_drop:N \l_tmpa_box }
      \@@_put_box_in_flow_i:
  }
```

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).

```
\cs_new_protected:Npn \@@_put_box_in_flow_i:
  {
    \pgfpicture
      \@@_qpoint:n { row - 1 }
      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
      \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
      \dim_gadd:Nn \g_tmpa_dim \pgf@y
      \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).

```
      \str_if_in:NnTF \l_@@_baseline_tl { line- }
        {
          \int_set:Nn \l_tmpa_int
            {
              \str_range:Nnn
                \l_@@_baseline_tl
                6
                { \tl_count:V \l_@@_baseline_tl }
            }
          \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
        }
        {
          \str_case:VnF \l_@@_baseline_tl
            {
              { t } { \int_set:Nn \l_tmpa_int 1 }
              { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
            }
            { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
          \bool_lazy_or:nnT
            { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
            { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
            {
              \@@_error:n { bad~value~for~baseline }
              \int_set:Nn \l_tmpa_int 1
            }
          \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
```

We take into account the position of the mathematical axis.

```
          \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
        }
      \dim_gsub:Nn \g_tmpa_dim \pgf@y
```

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

```
    \endpgfpicture
    \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
    \box_use_drop:N \l_tmpa_box
  }
```

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).

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

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

```
    \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
      {
        \box_set_wd:Nn \l_@@_the_array_box
          { \box_wd:N \l_@@_the_array_box - \arraycolsep }
      }
```

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

```
    \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
```

The \hbox avoids that the pgfpicture inside \@@_draw_blocks adds a extra vertical space before the notes.

```
    \hbox
      {
        \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.

```
        \@@_create_extra_nodes:
        \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
      }
    \bool_lazy_or:nnT
      { \int_compare_p:nNn \c@tabularnote > 0 }
      { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
      \@@_insert_tabularnotes:
    \end { minipage }
  }
```

```
\cs_new_protected:Npn \@@_insert_tabularnotes:
  {
    \skip_vertical:N 0.65ex
```

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

```
    \group_begin:
    \l_@@_notes_code_before_tl
    \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.

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

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

```
            \par
          }
          {
            \tabularnotes
              \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
            \endtabularnotes
          }
      }
    \unskip
    \group_end:
    \bool_if:NT \l_@@_notes_bottomrule_bool
      {
        \bool_if:NTF \c_@@_booktabs_loaded_bool
          {
```

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

```
            \skip_vertical:N \aboverulesep
```

\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.

```
            { \CT@arc@ \hrule height \heavyrulewidth }
          }
          { \@@_error:n { bottomrule~without~booktabs } }
      }
    \l_@@_notes_code_after_tl
    \seq_gclear:N \g_@@_tabularnotes_seq
    \int_gzero:N \c@tabularnote
  }
```

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.

```
\cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
  {
    \pgfpicture
      \@@_qpoint:n { row - 1 }
      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
      \@@_qpoint:n { row - \int_use:N \c@iRow - base }
      \dim_gsub:Nn \g_tmpa_dim \pgf@y
    \endpgfpicture
    \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
    \int_compare:nNnT \l_@@_first_row_int = 0
      {
        \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
        \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
      }
    \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
  }
```

Now, the general case.

```
\cs_new_protected:Npn \@@_use_arraybox_with_notes:
  {
```

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

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

```
    \pgfpicture
    \@@_qpoint:n { row - 1 }
    \dim_gset_eq:NN \g_tmpa_dim \pgf@y
    \str_if_in:NnTF \l_@@_baseline_tl { line- }
      {
        \int_set:Nn \l_tmpa_int
          {
            \str_range:Nnn
              \l_@@_baseline_tl
              6
              { \tl_count:V \l_@@_baseline_tl }
          }
        \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
      }
      {
        \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
        \bool_lazy_or:nnT
          { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
          { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
          {
            \@@_error:n { bad~value~for~baseline }
            \int_set:Nn \l_tmpa_int 1
          }
        \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
      }
    \dim_gsub:Nn \g_tmpa_dim \pgf@y
    \endpgfpicture
    \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
    \int_compare:nNnT \l_@@_first_row_int = 0
      {
        \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
        \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
      }
    \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
  }
```

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.

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

We will compute the real width of both delimiters used.

```
    \dim_zero_new:N \l_@@_real_left_delim_dim
    \dim_zero_new:N \l_@@_real_right_delim_dim
    \hbox_set:Nn \l_tmpb_box
      {
        \c_math_toggle_token
        \left #1
        \vcenter
          {
            \vbox_to_ht:nn
              { \box_ht_plus_dp:N \l_tmpa_box }
              { }
          }
        \right .
        \c_math_toggle_token
      }
    \dim_set:Nn \l_@@_real_left_delim_dim
      { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
    \hbox_set:Nn \l_tmpb_box
      {
        \c_math_toggle_token
        \left .
        \vbox_to_ht:nn
          { \box_ht_plus_dp:N \l_tmpa_box }
          { }
        \right #2
        \c_math_toggle_token
      }
    \dim_set:Nn \l_@@_real_right_delim_dim
      { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
    \skip_horizontal:N  \l_@@_left_delim_dim
    \skip_horizontal:N -\l_@@_real_left_delim_dim
    \@@_put_box_in_flow:
    \skip_horizontal:N \l_@@_right_delim_dim
    \skip_horizontal:N -\l_@@_real_right_delim_dim
  }
```

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).

```
\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).

```
  {
    \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).

```
      { \exp_args:NV \@@_array: \g_@@_preamble_tl }
  }
  {
    \@@_create_col_nodes:
    \endarray
  }
```

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).

```
\NewDocumentEnvironment { @@-light-syntax } { b }
  {
```

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.

```
    \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
    \tl_map_inline:nn { #1 }
      {
        \str_if_eq:nnT { ##1 } { & }
          { \@@_fatal:n { ampersand~in~light-syntax } }
        \str_if_eq:nnT { ##1 } { \\ }
          { \@@_fatal:n { double-backslash~in~light-syntax } }
      }
```

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 catched 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.

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

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.

```
  { }
```

```
\cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
  {
    \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).

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

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

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

```
    \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.

```
    \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
    \@@_line_with_light_syntax_i:V \l_tmpa_tl
    \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
    \@@_create_col_nodes:
    \endarray
  }
\cs_new_protected:Npn \@@_line_with_light_syntax:n #1
  { \tl_if_empty:nF { #1 } { \\ \@@_line_with_light_syntax_i:n { #1 } } }
\cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
  {
    \seq_gclear_new:N \g_@@_cells_seq
    \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
    \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
    \l_tmpa_tl
    \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
  }
\cs_generate_variant:Nn \@@_line_with_light_syntax_i:n { V }
```

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

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

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

```
    \end { #2 }
  }
```

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).

```
\cs_new:Npn \@@_create_col_nodes:
  {
    \crcr
    \int_compare:nNnT \l_@@_first_col_int = 0
      {
        \omit
        \hbox_overlap_left:n
          {
            \bool_if:NT \l_@@_code_before_bool
              { \pgfsys@markposition { \@@_env: - col - 0 } }
            \pgfpicture
            \pgfrememberpicturepositiononpagetrue
            \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
            \str_if_empty:NF \l_@@_name_str
              { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
            \endpgfpicture
            \skip_horizontal:N 2\col@sep
            \skip_horizontal:N \g_@@_width_first_col_dim
          }
        &
      }
    \omit
```

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

```
    \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`).

```
    \int_compare:nNnTF \l_@@_first_col_int = 0
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \hbox
              {
                \skip_horizontal:N -0.5\arrayrulewidth
                \pgfsys@markposition { \@@_env: - col - 1 }
                \skip_horizontal:N 0.5\arrayrulewidth
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 1 }
          { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
        \endpgfpicture
      }
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \hbox
              {
                \skip_horizontal:N 0.5\arrayrulewidth
                \pgfsys@markposition { \@@_env: - col - 1 }
                \skip_horizontal:N -0.5\arrayrulewidth
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 1 }
          { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
        \endpgfpicture
      }
```

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.

```
    \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
    \bool_if:NF \l_@@_auto_columns_width_bool
      { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
      {
        \bool_lazy_and:nnTF
          \l_@@_auto_columns_width_bool
          { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
          { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
          { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
        \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
      }
    \skip_horizontal:N \g_tmpa_skip
    \hbox
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \hbox
              {
                \skip_horizontal:N -0.5\arrayrulewidth
                \pgfsys@markposition { \@@_env: - col - 2 }
                \skip_horizontal:N 0.5\arrayrulewidth
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 2 }
          { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
        \endpgfpicture
      }
```

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.

```
\int_gset:Nn \g_tmpa_int 1
\bool_if:NTF \g_@@_last_col_found_bool
  { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
  { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
  {
    &
    \omit
    \int_gincr:N \g_tmpa_int
```

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

```
    \skip_horizontal:N \g_tmpa_skip
    \bool_if:NT \l_@@_code_before_bool
      {
        \hbox
          {
            \skip_horizontal:N -0.5\arrayrulewidth
            \pgfsys@markposition
              { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
            \skip_horizontal:N 0.5\arrayrulewidth
          }
      }
```

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

```
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
        { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
      \str_if_empty:NF \l_@@_name_str
        {
          \pgfnodealias
            { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
            { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
        }
    \endpgfpicture
  }
```

```
    &
    \omit
```

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

```
      \int_compare:nNnT \g_@@_col_total_int = 1
        { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
      \skip_horizontal:N \g_tmpa_skip
      \int_gincr:N \g_tmpa_int
      \bool_lazy_all:nT
        {
          \l_@@_NiceArray_bool
          { \bool_not_p:n \l_@@_NiceTabular_bool }
          { \clist_if_empty_p:N \l_@@_vlines_clist }
          { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
          { ! \l_@@_bar_at_end_of_pream_bool }
        }
        { \skip_horizontal:N -\col@sep }
      \bool_if:NT \l_@@_code_before_bool
        {
          \hbox
            {
              \skip_horizontal:N -0.5\arrayrulewidth
```

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

```
              \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
                { \skip_horizontal:N -\arraycolsep }
              \pgfsys@markposition
                { \@@_env: - col - \int_eval:n {
                  \g_tmpa_int + 1 } }
              \skip_horizontal:N 0.5\arrayrulewidth
              \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
                { \skip_horizontal:N \arraycolsep }
            }
        }
      \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
          {
            \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
              {
                \pgfpoint
                  { - 0.5 \arrayrulewidth - \arraycolsep }
                  \c_zero_dim
              }
              { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
          }
        \str_if_empty:NF \l_@@_name_str
          {
            \pgfnodealias
              { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
              { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
          }
      \endpgfpicture
```

```
    \bool_if:NT \g_@@_last_col_found_bool
      {
        \hbox_overlap_right:n
          {
            \skip_horizontal:N \g_@@_width_last_col_dim
            \bool_if:NT \l_@@_code_before_bool
              {
                \pgfsys@markposition
                  { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
              }
            \pgfpicture
            \pgfrememberpicturepositiononpagetrue
            \pgfcoordinate
              { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
              \pgfpointorigin
            \str_if_empty:NF \l_@@_name_str
              {
                \pgfnodealias
                  {
                     \l_@@_name_str - col
                     - \int_eval:n { \g_@@_col_total_int + 1 }
                  }
                  { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
              }
            \endpgfpicture
          }
      }
    \cr
  }
```

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

```
\tl_const:Nn \c_@@_preamble_first_col_tl
  {
    >
      {
```

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

```
        \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
        \bool_gset_true:N \g_@@_after_col_zero_bool
        \@@_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.

```
        \hbox_set:Nw \l_@@_cell_box
        \@@_math_toggle_token:
        \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”.

```
        \bool_lazy_and:nnT
          { \int_compare_p:nNn \c@iRow > 0 }
          {
            \bool_lazy_or_p:nn
              { \int_compare_p:nNn \l_@@_last_row_int < 0 }
              { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
          }
          {
            \l_@@_code_for_first_col_tl
            \xglobal \colorlet { nicematrix-first-col } { . }
          }
      }
```

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`.

```
    l
    <
      {
        \@@_math_toggle_token:
        \hbox_set_end:
        \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
        \@@_adjust_size_box:
        \@@_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.

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

```
        \hbox_overlap_left:n
          {
            \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
              \@@_node_for_cell:
              { \box_use_drop:N \l_@@_cell_box }
            \skip_horizontal:N \l_@@_left_delim_dim
            \skip_horizontal:N \l_@@_left_margin_dim
            \skip_horizontal:N \l_@@_extra_left_margin_dim
          }
        \bool_gset_false:N \g_@@_empty_cell_bool
        \skip_horizontal:N -2\col@sep
      }
  }
```

Here is the preamble for the “last column” (if the user uses the key `last-col`).

```
\tl_const:Nn \c_@@_preamble_last_col_tl
  {
    >
      {
```

At the beginning of the cell, we link `\CodeAfter` to a command which do begins with `\\` (whereas the standard version of `\CodeAfter` begins does not).

```
        \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```
        \bool_gset_true:N \g_@@_last_col_found_bool
        \int_gincr:N \c@jCol
        \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.

```
        \hbox_set:Nw \l_@@_cell_box
          \@@_math_toggle_token:
          \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”.

```
        \int_compare:nNnT \c@iRow > 0
          {
            \bool_lazy_or:nnT
              { \int_compare_p:nNn \l_@@_last_row_int < 0 }
              { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
              {
                \l_@@_code_for_last_col_tl
                \xglobal \colorlet { nicematrix-last-col } { . }
              }
          }
      }
    l
    <
      {
        \@@_math_toggle_token:
        \hbox_set_end:
        \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
        \@@_adjust_size_box:
        \@@_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.

```
          \dim_gset:Nn \g_@@_width_last_col_dim
            { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
          \skip_horizontal:N -2\col@sep
```

The content of the cell is inserted in an overlapping position.

```
          \hbox_overlap_right:n
            {
              \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
                {
                  \skip_horizontal:N \l_@@_right_delim_dim
                  \skip_horizontal:N \l_@@_right_margin_dim
                  \skip_horizontal:N \l_@@_extra_right_margin_dim
                  \@@_node_for_cell:
                }
            }
          \bool_gset_false:N \g_@@_empty_cell_bool
        }
    }
```

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.

```
\NewDocumentEnvironment { NiceArray } { }
  {
    \bool_set_true:N \l_@@_NiceArray_bool
    \str_if_empty:NT \g_@@_name_env_str
      { \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).

```
    \NiceArrayWithDelims . .
  }
  { \endNiceArrayWithDelims }
```

We create the variants of the environment {NiceArrayWithDelims}.

```
\cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
  {
    \NewDocumentEnvironment { #1 NiceArray } { }
      {
        \str_if_empty:NT \g_@@_name_env_str
          { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
        \@@_test_if_math_mode:
        \NiceArrayWithDelims #2 #3
      }
      { \endNiceArrayWithDelims }
  }
```

```
\@@_def_env:nnn p ( )
\@@_def_env:nnn b [ ]
\@@_def_env:nnn B \{ \}
\@@_def_env:nnn v | |
\@@_def_env:nnn V \| \|
```

The environment {NiceMatrix} and its variants

```
\cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
  {
    \bool_set_true:N \l_@@_Matrix_bool
    \use:c { #1 NiceArray }
      {
        *
          {
            \int_compare:nNnTF \l_@@_last_col_int < 0
              \c@MaxMatrixCols
              { \int_eval:n { \l_@@_last_col_int - 1 } }
          }
          { > \@@_cell_begin:w #2 < \@@_cell_end: }
      }
  }
\cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n e }
```

```
\clist_map_inline:nn { { } , p , b , B , v , V }
  {
    \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
      {
        \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
        \tl_set:Nn \l_@@_type_of_col_tl c
        \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
        \@@_begin_of_NiceMatrix:ne { #1 } \l_@@_type_of_col_tl
      }
      { \use:c { end #1 NiceArray } }
  }
```

The following command will be linked to \NotEmpty in the environments of nicematrix.

```
\cs_new_protected:Npn \@@_NotEmpty:
  { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

{NiceTabular}, {NiceTabularX} and {NiceTabular*}

```
\NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
  {
```

If the dimension \l_@@_width_dim is equal to 0 pt, that means that it has not be set by a previous use of \NiceMatrixOptions.

```
    \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
      { \dim_set_eq:NN \l_@@_width_dim \linewidth }
    \str_gset:Nn \g_@@_name_env_str { NiceTabular }
    \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
    \bool_set_true:N \l_@@_NiceTabular_bool
    \NiceArray { #2 }
  }
  { \endNiceArray }
```

```
\cs_set_protected:Npn \@@_newcolumntype #1
  {
    \cs_if_free:cT { NC @ find @ #1 }
      { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
    \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
    \peek_meaning:NTF [
      { \newcol@ #1 }
      { \newcol@ #1 [ 0 ] }
  }
```

```
\NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
  {
```

The following code prevents the expansion of the 'X' columns with the definition of that columns in tabularx (this would result in an error in {NiceTabularX}).

```
    \bool_if:NT \c_@@_tabularx_loaded_bool { \newcolumntype { X } { \@@_X } }
    \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
    \dim_zero_new:N \l_@@_width_dim
    \dim_set:Nn \l_@@_width_dim { #1 }
    \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
    \bool_set_true:N \l_@@_NiceTabular_bool
    \NiceArray { #3 }
  }
  { \endNiceArray }
```

```
\NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
  {
    \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
    \dim_set:Nn \l_@@_tabular_width_dim { #1 }
    \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
    \bool_set_true:N \l_@@_NiceTabular_bool
    \NiceArray { #3 }
  }
  { \endNiceArray }
```

After the construction of the array

```
\cs_new_protected:Npn \@@_after_array:
  {
    \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.

```
    \bool_if:NT \g_@@_last_col_found_bool
      { \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.

```
    \bool_if:NT \l_@@_last_col_without_value_bool
      { \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.

```
    \bool_if:NT \l_@@_last_row_without_value_bool
      { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

    \tl_gput_right:Nx \g_@@_aux_tl
      {
        \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
          {
            \int_use:N \l_@@_first_row_int ,
            \int_use:N \c@iRow ,
            \int_use:N \g_@@_row_total_int ,
            \int_use:N \l_@@_first_col_int ,
            \int_use:N \c@jCol ,
            \int_use:N \g_@@_col_total_int
          }
      }
```

We write also the potential content of \g_@@_pos_of_blocks_seq. It will be used to recreate the blocks with a name in the \CodeBefore and also if the command \rowcolors is used with the key respect-blocks).

```
    \seq_if_empty:NF \g_@@_pos_of_blocks_seq
```

```
      {
        \tl_gput_right:Nx \g_@@_aux_tl
          {
            \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
              { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
          }
      }
    \seq_if_empty:NF \g_@@_multicolumn_cells_seq
      {
        \tl_gput_right:Nx \g_@@_aux_tl
          {
            \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
              { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
            \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
              { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
          }
      }
```

Now, you create the diagonal nodes by using the `row` nodes and the `col` nodes.

```
    \@@_create_diag_nodes:
```

We create the aliases using `last` for the nodes of the cells in the last row and the last column.

```
    \pgfpicture
    \int_step_inline:nn \c@iRow
      {
        \pgfnodealias
          { \@@_env: - ##1 - last }
          { \@@_env: - ##1 - \int_use:N \c@jCol }
      }
    \int_step_inline:nn \c@jCol
      {
        \pgfnodealias
          { \@@_env: - last - ##1 }
          { \@@_env: - \int_use:N \c@iRow - ##1 }
      }
    \str_if_empty:NF \l_@@_name_str
      {
        \int_step_inline:nn \c@iRow
          {
            \pgfnodealias
              { \l_@@_name_str - ##1 - last }
              { \@@_env: - ##1 - \int_use:N \c@jCol }
          }
        \int_step_inline:nn \c@jCol
          {
            \pgfnodealias
              { \l_@@_name_str - last - ##1 }
              { \@@_env: - \int_use:N \c@iRow - ##1 }
          }
      }
    \endpgfpicture
```

By default, the diagonal lines will be parallelized[68]. 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.

```
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \int_gzero_new:N \g_@@_ddots_int
        \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`

[68] It's possible to use the option `parallelize-diags` to disable this parallelization.

diagonals parallel to the first one. Similarly \g_@@_delta_x_two_dim and \g_@@_delta_y_two_dim are the Δ_x and Δ_y of the first \Iddots diagonal.

```
      \dim_gzero_new:N \g_@@_delta_x_one_dim
      \dim_gzero_new:N \g_@@_delta_y_one_dim
      \dim_gzero_new:N \g_@@_delta_x_two_dim
      \dim_gzero_new:N \g_@@_delta_y_two_dim
     }
    \int_zero_new:N \l_@@_initial_i_int
    \int_zero_new:N \l_@@_initial_j_int
    \int_zero_new:N \l_@@_final_i_int
    \int_zero_new:N \l_@@_final_j_int
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
```

If the option small is used, the values \l_@@_radius_dim and \l_@@_inter_dots_dim (used to draw the dotted lines created by \hdottedline and \vdottedline and also for all the other dotted lines when line-style is equal to standard, which is the initial value) are changed.

```
    \bool_if:NT \l_@@_small_bool
     {
      \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
      \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }
```

The dimension \l_@@_xdots_shorten_dim corresponds to the option xdots/shorten available to the user. That’s why we give a new value according to the current value, and not an absolute value.

```
      \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
     }
```

Now, we actually draw the dotted lines (specified by \Cdots, \Vdots, etc.).

```
    \@@_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.

```
    \@@_compute_corners:
```

The sequence \g_@@_pos_of_blocks_seq must be “adjusted” (for the case where the user have written something like \Block{1-*}).

```
    \@@_adjust_pos_of_blocks_seq:
    \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
    \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

Now, the internal code-after and then, the \CodeAfter.

```
    \bool_if:NT \c_@@_tikz_loaded_bool
     {
      \tikzset
       {
        every~picture / .style =
         {
          overlay ,
          remember~picture ,
          name~prefix = \@@_env: -
         }
       }
     }
    \cs_set_eq:NN \ialign \@@_old_ialign:
    \cs_set_eq:NN \SubMatrix \@@_SubMatrix
    \cs_set_eq:NN \UnderBrace \@@_UnderBrace
    \cs_set_eq:NN \OverBrace \@@_OverBrace
    \cs_set_eq:NN \line \@@_line
    \g_@@_internal_code_after_tl
    \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.

```
    \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).

```
    \seq_gclear:N \g_@@_submatrix_names_seq
```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an "argument" between square brackets of the options, we extract and treat that potential "argument" with the command `\@@_CodeAfter_keys:`.

```
    \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
    \scan_stop:
    \tl_gclear:N \g_nicematrix_code_after_tl
    \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.

```
    \tl_if_empty:NF \g_nicematrix_code_before_tl
      {
```

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`.

```
        \cs_set_protected:Npn \rectanglecolor { }
        \cs_set_protected:Npn \columncolor { }
        \tl_gput_right:Nx \g_@@_aux_tl
          {
            \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
              { \exp_not:V \g_nicematrix_code_before_tl }
          }
        \bool_set_true:N \l_@@_code_before_bool
      }

    \str_gclear:N \g_@@_name_env_str
    \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array[69]. 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.

```
    \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
  }
```

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`.

```
\NewDocumentCommand \@@_CodeAfter_keys: { O { } }
  { \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).

[69] e.g. `\color[rgb]{0.5,0.5,0}`

```
\cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
  {
    \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
      { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
  }
```

The following command must *not* be protected.

```
\cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
  {
    { #1 }
    { #2 }
    {
      \int_compare:nNnTF { #3 } > { 99 }
        { \int_use:N \c@iRow }
        { #3 }
    }
    {
      \int_compare:nNnTF { #4 } > { 99 }
        { \int_use:N \c@jCol }
        { #4 }
    }
    { #5 }
  }
```

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).

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \cs_new_protected:Npx \@@_draw_dotted_lines:
      {
        \c_@@_pgfortikzpicture_tl
        \@@_draw_dotted_lines_i:
        \c_@@_endpgfortikzpicture_tl
      }
  }
```

The following command *must* be protected because it will appear in the construction of the command `\@@_draw_dotted_lines:`.

```
\cs_new_protected:Npn \@@_draw_dotted_lines_i:
  {
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \g_@@_HVdotsfor_lines_tl
    \g_@@_Vdots_lines_tl
    \g_@@_Ddots_lines_tl
    \g_@@_Iddots_lines_tl
    \g_@@_Cdots_lines_tl
    \g_@@_Ldots_lines_tl
  }
```

```
\cs_new_protected:Npn \@@_restore_iRow_jCol:
  {
    \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
    \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
  }
```

We define a new PGF shape for the diag nodes because we want to provide a anchor called `.5` for those nodes.

```
\pgfdeclareshape { @@_diag_node }
  {
    \savedanchor { \five }
      {
        \dim_gset_eq:NN \pgf@x \l_tmpa_dim
        \dim_gset_eq:NN \pgf@y \l_tmpb_dim
      }
    \anchor { 5 } { \five }
    \anchor { center } { \pgfpointorigin }
  }
```

The following command creates the diagonal nodes (in fact, if the matrix is not a square matrix, not all the nodes are on the diagonal).

```
\cs_new_protected:Npn \@@_create_diag_nodes:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
      {
        \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
        \dim_set_eq:NN \l_tmpa_dim \pgf@x
        \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
        \dim_set_eq:NN \l_tmpb_dim \pgf@y
        \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
        \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
        \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 `@à_diag_node`) that we will construct.

```
        \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
        \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
        \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
      }
```

Now, the last node. Of course, that is only a `coordinate` because there is not `.5` anchor for that node.

```
    \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
    \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
    \pgfcoordinate
      { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
    \pgfnodealias
      { \@@_env: - last }
      { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
    \str_if_empty:NF \l_@@_name_str
      {
        \pgfnodealias
          { \l_@@_name_str - \int_use:N \l_tmpa_int }
          { \@@_env: - \int_use:N \l_tmpa_int }
        \pgfnodealias
          { \l_@@_name_str - last }
          { \@@_env: - last }
      }
    \endpgfpicture
  }
```

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.

$$\begin{pmatrix} a+b+c & a+b & a \\ a \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & & \\ a & a+b & a+b+c \end{pmatrix}$$

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.

```
\cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
  {
```

First, we declare the current cell as “dotted” because we forbide intersections of dotted lines.

```
    \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
    \int_set:Nn \l_@@_initial_i_int { #1 }
    \int_set:Nn \l_@@_initial_j_int { #2 }
    \int_set:Nn \l_@@_final_i_int { #1 }
    \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.

```
    \bool_set_false:N \l_@@_stop_loop_bool
    \bool_do_until:Nn \l_@@_stop_loop_bool
      {
        \int_add:Nn \l_@@_final_i_int { #3 }
        \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
        \bool_set_false:N \l_@@_final_open_bool
        \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
          {
            \int_compare:nNnTF { #3 } = 1
              { \bool_set_true:N \l_@@_final_open_bool }
              {
                \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
                  { \bool_set_true:N \l_@@_final_open_bool }
              }
          }
          {
            \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
              {
                \int_compare:nNnT { #4 } = { -1 }
                  { \bool_set_true:N \l_@@_final_open_bool }
              }
              {
                \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
                  {
                    \int_compare:nNnT { #4 } = 1
                      { \bool_set_true:N \l_@@_final_open_bool }
                  }
              }
          }
        \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.

```
          {
```

We do a step backwards.

```
            \int_sub:Nn \l_@@_final_i_int { #3 }
            \int_sub:Nn \l_@@_final_j_int { #4 }
            \bool_set_true:N \l_@@_stop_loop_bool
          }
```

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`.

```
          {
            \cs_if_exist:cTF
              {
                @@ _ dotted _
                \int_use:N \l_@@_final_i_int -
                \int_use:N \l_@@_final_j_int
              }
              {
                \int_sub:Nn \l_@@_final_i_int { #3 }
                \int_sub:Nn \l_@@_final_j_int { #4 }
                \bool_set_true:N \l_@@_final_open_bool
                \bool_set_true:N \l_@@_stop_loop_bool
              }
              {
                \cs_if_exist:cTF
                  {
                    pgf @ sh @ ns @ \@@_env:
                    - \int_use:N \l_@@_final_i_int
                    - \int_use:N \l_@@_final_j_int
                  }
                  { \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.

```
                  {
                    \cs_set:cpn
                      {
                        @@ _ dotted _
                        \int_use:N \l_@@_final_i_int -
                        \int_use:N \l_@@_final_j_int
                      }
                      { }
                  }
              }
          }
      }
```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programmation is similar to the previous one.

```
    \bool_set_false:N \l_@@_stop_loop_bool
```

```
\bool_do_until:Nn \l_@@_stop_loop_bool
  {
    \int_sub:Nn \l_@@_initial_i_int { #3 }
    \int_sub:Nn \l_@@_initial_j_int { #4 }
    \bool_set_false:N \l_@@_initial_open_bool
    \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
      {
        \int_compare:nNnTF { #3 } = 1
          { \bool_set_true:N \l_@@_initial_open_bool }
          {
            \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
              { \bool_set_true:N \l_@@_initial_open_bool }
          }
      }
      {
        \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
          {
            \int_compare:nNnT { #4 } = 1
              { \bool_set_true:N \l_@@_initial_open_bool }
          }
          {
            \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
              {
                \int_compare:nNnT { #4 } = { -1 }
                  { \bool_set_true:N \l_@@_initial_open_bool }
              }
          }
      }
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \int_add:Nn \l_@@_initial_i_int { #3 }
        \int_add:Nn \l_@@_initial_j_int { #4 }
        \bool_set_true:N \l_@@_stop_loop_bool
      }
      {
        \cs_if_exist:cTF
          {
            @@ _ dotted _
            \int_use:N \l_@@_initial_i_int -
            \int_use:N \l_@@_initial_j_int
          }
          {
            \int_add:Nn \l_@@_initial_i_int { #3 }
            \int_add:Nn \l_@@_initial_j_int { #4 }
            \bool_set_true:N \l_@@_initial_open_bool
            \bool_set_true:N \l_@@_stop_loop_bool
          }
          {
            \cs_if_exist:cTF
              {
                pgf @ sh @ ns @ \@@_env:
                - \int_use:N \l_@@_initial_i_int
                - \int_use:N \l_@@_initial_j_int
              }
              { \bool_set_true:N \l_@@_stop_loop_bool }
              {
                \cs_set:cpn
                  {
                    @@ _ dotted _
                    \int_use:N \l_@@_initial_i_int -
                    \int_use:N \l_@@_initial_j_int
                  }
                  { }
                  }
              }
          }
      }
```

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.

```
    \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
      {
        { \int_use:N \l_@@_initial_i_int }
```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```
        { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
        { \int_use:N \l_@@_final_i_int }
        { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
        { } % for the name of the block
      }
  }
```

The following commmand (*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).

```
\cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
  {
    \int_set:Nn \l_@@_row_min_int 1
    \int_set:Nn \l_@@_col_min_int 1
    \int_set_eq:NN \l_@@_row_max_int \c@iRow
    \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`.

```
    \seq_map_inline:Nn \g_@@_submatrix_seq
      { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
  }
```

`#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.

```
\cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
  {
    \bool_if:nT
      {
           \int_compare_p:n { #3 <= #1 }
        && \int_compare_p:n { #1 <= #5 }
        && \int_compare_p:n { #4 <= #2 }
        && \int_compare_p:n { #2 <= #6 }
      }
      {
        \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
        \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
        \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
        \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
      }
  }
```

```
\cs_new_protected:Npn \@@_set_initial_coords:
  {
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
  }
\cs_new_protected:Npn \@@_set_final_coords:
  {
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
  }
\cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
  {
    \pgfpointanchor
      {
        \@@_env:
        - \int_use:N \l_@@_initial_i_int
        - \int_use:N \l_@@_initial_j_int
      }
      { #1 }
    \@@_set_initial_coords:
  }
\cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
  {
    \pgfpointanchor
      {
        \@@_env:
        - \int_use:N \l_@@_final_i_int
        - \int_use:N \l_@@_final_j_int
      }
      { #1 }
    \@@_set_final_coords:
  }
```

```
\cs_new_protected:Npn \@@_open_x_initial_dim:
  {
    \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
    \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
          {
            \pgfpointanchor
              { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
              { west }
            \dim_set:Nn \l_@@_x_initial_dim
              { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
          }
      }
```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```
    \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
      {
        \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
        \dim_add:Nn \l_@@_x_initial_dim \col@sep
      }
  }
```

```
\cs_new_protected:Npn \@@_open_x_final_dim:
  {
    \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
    \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
          {
            \pgfpointanchor
              { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
              { east }
            \dim_set:Nn \l_@@_x_final_dim
              { \dim_max:nn \l_@@_x_final_dim \pgf@x }
          }
        }
```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```
    \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
      {
        \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
        \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
        \dim_sub:Nn \l_@@_x_final_dim \col@sep
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_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.

```
        \group_begin:
          \int_compare:nNnTF { #1 } = 0
            { \color { nicematrix-first-row } }
            {
```

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.

```
              \int_compare:nNnT { #1 } = \l_@@_last_row_int
                { \color { nicematrix-last-row } }
            }
          \keys_set:nn { NiceMatrix / xdots } { #3 }
          \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
          \@@_actually_draw_Ldots:
        \group_end:
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_actually_draw_Ldots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_open_x_initial_dim:
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
        \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
      }
      { \@@_set_initial_coords_from_anchor:n { base~east } }
    \bool_if:NTF \l_@@_final_open_bool
```

```
      {
        \@@_open_x_final_dim:
        \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
        \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
      }
      { \@@_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 texte. Of course, maybe we should not do that when the option `line-style` is used (?).

```
    \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
    \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
    \@@_draw_line:
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_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.

```
        \group_begin:
          \int_compare:nNnTF { #1 } = 0
            { \color { nicematrix-first-row } }
            {
```

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.

```
              \int_compare:nNnT { #1 } = \l_@@_last_row_int
                { \color { nicematrix-last-row } }
            }
          \keys_set:nn { NiceMatrix / xdots } { #3 }
          \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
          \@@_actually_draw_Cdots:
        \group_end:
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_actually_draw_Cdots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      { \@@_open_x_initial_dim: }
      { \@@_set_initial_coords_from_anchor:n { mid~east } }
    \bool_if:NTF \l_@@_final_open_bool
      { \@@_open_x_final_dim: }
      { \@@_set_final_coords_from_anchor:n { mid~west } }
    \bool_lazy_and:nnTF
```

```
      \l_@@_initial_open_bool
      \l_@@_final_open_bool
      {
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
        \dim_set_eq:NN \l_tmpa_dim \pgf@y
        \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
        \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
        \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
      }
      {
        \bool_if:NT \l_@@_initial_open_bool
          { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
        \bool_if:NT \l_@@_final_open_bool
          { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
      }
    \@@_draw_line:
  }
\cs_new_protected:Npn \@@_open_y_initial_dim:
  {
    \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
    \dim_set:Nn \l_@@_y_initial_dim
      { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
    \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
          {
            \pgfpointanchor
              { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
              { north }
            \dim_set:Nn \l_@@_y_initial_dim
              { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
          }
      }
  }
\cs_new_protected:Npn \@@_open_y_final_dim:
  {
    \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
    \dim_set:Nn \l_@@_y_final_dim
      { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
    \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
          {
            \pgfpointanchor
              { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
              { south }
            \dim_set:Nn \l_@@_y_final_dim
              { \dim_min:nn \l_@@_y_final_dim \pgf@y }
          }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_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.

```
        \group_begin:
          \int_compare:nNnTF { #2 } = 0
            { \color { nicematrix-first-col } }
            {
              \int_compare:nNnT { #2 } = \l_@@_last_col_int
                { \color { nicematrix-last-col } }
            }
          \keys_set:nn { NiceMatrix / xdots } { #3 }
          \tl_if_empty:VF \l_@@_xdots_color_tl
            { \color { \l_@@_xdots_color_tl } }
          \@@_actually_draw_Vdots:
        \group_end:
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_actually_draw_Vdots:
  {
```

The boolean `\l_tmpa_bool` indicates whether the column is of type `l` or may be considered as if.

```
    \bool_set_false:N \l_tmpa_bool
```

First the case when the line is closed on both ends.

```
    \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
      {
        \@@_set_initial_coords_from_anchor:n { south~west }
        \@@_set_final_coords_from_anchor:n { north~west }
        \bool_set:Nn \l_tmpa_bool
          { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
      }
```

Now, we try to determine whether the column is of type `c` or may be considered as if.

```
    \bool_if:NTF \l_@@_initial_open_bool
      \@@_open_y_initial_dim:
      { \@@_set_initial_coords_from_anchor:n { south } }
    \bool_if:NTF \l_@@_final_open_bool
      \@@_open_y_final_dim:
      { \@@_set_final_coords_from_anchor:n { north } }
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \bool_if:NTF \l_@@_final_open_bool
          {
            \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
            \dim_set_eq:NN \l_tmpa_dim \pgf@x
            \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
            \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
            \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.

```
                \int_compare:nNnT \l_@@_last_col_int > { -2 }
                  {
                    \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
                      {
                        \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
                        \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
                        \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
                        \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
                      }
                  }
              }
              { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
          }
          {
            \bool_if:NTF \l_@@_final_open_bool
              { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
              {
```

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.

```
                \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
                  {
                    \dim_set:Nn \l_@@_x_initial_dim
                      {
                        \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
                          \l_@@_x_initial_dim \l_@@_x_final_dim
                      }
                    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
                  }
              }
          }
        \@@_draw_line:
      }
```

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.

```
\cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_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.

```
        \group_begin:
          \keys_set:nn { NiceMatrix / xdots } { #3 }
          \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
          \@@_actually_draw_Ddots:
        \group_end:
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_actually_draw_Ddots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_open_y_initial_dim:
        \@@_open_x_initial_dim:
      }
      { \@@_set_initial_coords_from_anchor:n { south~east } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_open_x_final_dim:
        \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
      }
      { \@@_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.

```
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \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).

```
        \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.

```
          {
            \dim_gset:Nn \g_@@_delta_x_one_dim
              { \l_@@_x_final_dim - \l_@@_x_initial_dim }
            \dim_gset:Nn \g_@@_delta_y_one_dim
              { \l_@@_y_final_dim - \l_@@_y_initial_dim }
          }
```

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`.

```
          {
            \dim_set:Nn \l_@@_y_final_dim
              {
                \l_@@_y_initial_dim +
                ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
                \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
              }
          }
      }
    \@@_draw_line:
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
```

```
      {
        \@@_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.

```
        \group_begin:
          \keys_set:nn { NiceMatrix / xdots } { #3 }
          \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
          \@@_actually_draw_Iddots:
        \group_end:
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_actually_draw_Iddots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_open_y_initial_dim:
        \@@_open_x_initial_dim:
      }
      { \@@_set_initial_coords_from_anchor:n { south~west } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_open_y_final_dim:
        \@@_open_x_final_dim:
      }
      { \@@_set_final_coords_from_anchor:n { north~east } }
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \int_gincr:N \g_@@_iddots_int
        \int_compare:nNnTF \g_@@_iddots_int = 1
          {
            \dim_gset:Nn \g_@@_delta_x_two_dim
              { \l_@@_x_final_dim - \l_@@_x_initial_dim }
            \dim_gset:Nn \g_@@_delta_y_two_dim
              { \l_@@_y_final_dim - \l_@@_y_initial_dim }
          }
          {
            \dim_set:Nn \l_@@_y_final_dim
              {
                \l_@@_y_initial_dim +
                ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
                \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
              }
          }
      }
    \@@_draw_line:
  }
```

The actual instructions for drawing the dotted lines with Tikz

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```
\cs_new_protected:Npn \@@_draw_line:
  {
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \bool_lazy_or:nnTF
    { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
```

The boolean `\l_@@_dotted_bool` is raised for the rules specified by either `\hdottedline` or `:` (or the letter specified by `letter-for-dotted-lines`) in the preamble of the array.

```
    \l_@@_dotted_bool
      \@@_draw_standard_dotted_line:
      \@@_draw_unstandard_dotted_line:
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
  {
    \begin { scope }
    \@@_draw_unstandard_dotted_line:o
      { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
  }
```

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 diredtly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```
\cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
  {
    \@@_draw_unstandard_dotted_line:nVV
      { #1 }
      \l_@@_xdots_up_tl
      \l_@@_xdots_down_tl
  }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

\cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
  {
    \draw
      [
        #1 ,
        shorten~> = \l_@@_xdots_shorten_dim ,
        shorten~< = \l_@@_xdots_shorten_dim ,
      ]
          ( \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).

```
          -- node [ sloped , above ] { $ \scriptstyle #2 $ }
             node [ sloped , below ] { $ \scriptstyle #3 $ }
             ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
```

```
    \end { scope }
  }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnn { n V V }
```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real round dots).

```
\cs_new_protected:Npn \@@_draw_standard_dotted_line:
  {
    \bool_lazy_and:nnF
      { \tl_if_empty_p:N \l_@@_xdots_up_tl }
      { \tl_if_empty_p:N \l_@@_xdots_down_tl }
      {
        \pgfscope
        \pgftransformshift
          {
            \pgfpointlineattime { 0.5 }
              { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
              { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
          }
        \pgftransformrotate
          {
            \fp_eval:n
              {
                atand
                 (
                   \l_@@_y_final_dim - \l_@@_y_initial_dim ,
                   \l_@@_x_final_dim - \l_@@_x_initial_dim
                  )
              }
          }
        \pgfnode
          { rectangle }
          { south }
          {
            \c_math_toggle_token
            \scriptstyle \l_@@_xdots_up_tl
            \c_math_toggle_token
          }
          { }
          { \pgfusepath { } }
        \pgfnode
          { rectangle }
          { north }
          {
            \c_math_toggle_token
            \scriptstyle \l_@@_xdots_down_tl
            \c_math_toggle_token
          }
          { }
          { \pgfusepath { } }
        \endpgfscope
      }
    \group_begin:
```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```
      \dim_zero_new:N \l_@@_l_dim
      \dim_set:Nn \l_@@_l_dim
        {
          \fp_to_dim:n
            {
              sqrt
               (
                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
                    +
                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
               )
            }
        }
```

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.

```
      \bool_lazy_or:nnF
        { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
        { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
        \@@_draw_standard_dotted_line_i:
    \group_end:
  }
```

```
\dim_const:Nn \c_@@_max_l_dim { 50 cm }
```

```
\cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
  {
```

The number of dots will be `\l_tmpa_int + 1`.

```
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \bool_if:NTF \l_@@_final_open_bool
          {
            \int_set:Nn \l_tmpa_int
              { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
          }
          {
            \int_set:Nn \l_tmpa_int
              {
                \dim_ratio:nn
                  { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
                  \l_@@_inter_dots_dim
              }
          }
      }
      {
        \bool_if:NTF \l_@@_final_open_bool
          {
            \int_set:Nn \l_tmpa_int
              {
                \dim_ratio:nn
                  { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
                  \l_@@_inter_dots_dim
              }
          }
          {
            \int_set:Nn \l_tmpa_int
              {
                \dim_ratio:nn
                  { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim  }
                  \l_@@_inter_dots_dim
              }
          }
      }
```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```
    \dim_set:Nn \l_tmpa_dim
      {
        ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
        \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
      }
    \dim_set:Nn \l_tmpb_dim
      {
        ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
        \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
      }
```

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`.

```
    \int_set:Nn \l_tmpb_int
      {
        \bool_if:NTF \l_@@_initial_open_bool
          { \bool_if:NTF \l_@@_final_open_bool 1 0 }
          { \bool_if:NTF \l_@@_final_open_bool 2 1 }
      }
```

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.

```
    \dim_gadd:Nn \l_@@_x_initial_dim
      {
        ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
        \dim_ratio:nn
          { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
          { 2 \l_@@_l_dim }
        * \l_tmpb_int
      }
    \dim_gadd:Nn \l_@@_y_initial_dim
      {
        ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
        \dim_ratio:nn
          { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
          { 2 \l_@@_l_dim }
        * \l_tmpb_int
      }
    \pgf@relevantforpicturesizefalse
    \int_step_inline:nnn 0 \l_tmpa_int
      {
        \pgfpathcircle
          { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
          { \l_@@_radius_dim }
        \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
        \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
      }
    \pgfusepathqfill
  }
```

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 thats' why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use underscore, and, in that case, the catcode is 13 because underscore activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \tl_set:Nn \l_@@_argspec_tl { O { } E { _ ^ } { { } { } } }
    \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
    \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
      {
        \int_compare:nNnTF \c@jCol = 0
          { \@@_error:nn { in~first~col } \Ldots }
          {
            \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
              { \@@_error:nn { in~last~col } \Ldots }
              {
                \@@_instruction_of_type:nnn \c_false_bool { Ldots }
                  { #1 , down = #2 , up = #3 }
              }
          }
        \bool_if:NF \l_@@_nullify_dots_bool
          { \phantom { \ensuremath { \@@_old_ldots } } }
        \bool_gset_true:N \g_@@_empty_cell_bool
      }


    \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
      {
        \int_compare:nNnTF \c@jCol = 0
          { \@@_error:nn { in~first~col } \Cdots }
          {
            \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
              { \@@_error:nn { in~last~col } \Cdots }
              {
                \@@_instruction_of_type:nnn \c_false_bool { Cdots }
                  { #1 , down = #2 , up = #3 }
              }
          }
        \bool_if:NF \l_@@_nullify_dots_bool
          { \phantom { \ensuremath { \@@_old_cdots } } }
        \bool_gset_true:N \g_@@_empty_cell_bool
      }


    \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
      {
        \int_compare:nNnTF \c@iRow = 0
          { \@@_error:nn { in~first~row } \Vdots }
          {
            \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
              { \@@_error:nn { in~last~row } \Vdots }
              {
                \@@_instruction_of_type:nnn \c_false_bool { Vdots }
                  { #1 , down = #2 , up = #3 }
              }
          }
        \bool_if:NF \l_@@_nullify_dots_bool
          { \phantom { \ensuremath { \@@_old_vdots } } }
        \bool_gset_true:N \g_@@_empty_cell_bool
      }


    \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
      {
        \int_case:nnF \c@iRow
          {
            0                  { \@@_error:nn { in~first~row } \Ddots }
            \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
```

```
     }
     {
      \int_case:nnF \c@jCol
       {
        0                  { \@@_error:nn { in~first~col } \Ddots }
        \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
       }
       {
        \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
        \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
         { #1 , down = #2 , up = #3 }
       }

     }
    \bool_if:NF \l_@@_nullify_dots_bool
     { \phantom { \ensuremath { \@@_old_ddots } } }
    \bool_gset_true:N \g_@@_empty_cell_bool
   }
```

```
  \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
   {
    \int_case:nnF \c@iRow
     {
      0                  { \@@_error:nn { in~first~row } \Iddots }
      \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
     }
     {
      \int_case:nnF \c@jCol
       {
        0                  { \@@_error:nn { in~first~col } \Iddots }
        \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
       }
       {
        \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
        \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
         { #1 , down = #2 , up = #3 }
       }
     }
    \bool_if:NF \l_@@_nullify_dots_bool
     { \phantom { \ensuremath { \@@_old_iddots } } }
    \bool_gset_true:N \g_@@_empty_cell_bool
   }
 }
```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```
\keys_define:nn { NiceMatrix / Ddots }
 {
  draw-first .bool_set:N = \l_@@_draw_first_bool ,
  draw-first .default:n = true ,
  draw-first .value_forbidden:n = true
 }
```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```
\cs_new_protected:Npn \@@_Hspace:
 {
  \bool_gset_true:N \g_@@_empty_cell_bool
  \hspace
 }
```

In the environments of nicematrix, the command \multicolumn is redefined. We will patch the environment {tabular} to go back to the previous value of \multicolumn.

```
\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.

```
\cs_new:Npn \@@_Hdotsfor:
  {
    \bool_lazy_and:nnTF
      { \int_compare_p:nNn \c@jCol = 0 }
      { \int_compare_p:nNn \l_@@_first_col_int = 0 }
      {
        \bool_if:NTF \g_@@_after_col_zero_bool
          {
            \multicolumn { 1 } { c } { }
            \@@_Hdotsfor_i
          }
          { \@@_fatal:n { Hdotsfor~in~col~0 } }
      }
      {
        \multicolumn { 1 } { c } { }
        \@@_Hdotsfor_i
      }
  }
```

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:).

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \tl_set:Nn \l_@@_argspec_tl { O { } m O { } E { _ ^ } { { } { } } }
    \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.

```
    \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
      {
        \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
          {
            \@@_Hdotsfor:nnnn
              { \int_use:N \c@iRow }
              { \int_use:N \c@jCol }
              { #2 }
              {
                #1 , #3 ,
                down = \exp_not:n { #4 } ,
                up = \exp_not:n { #5 }
              }
          }
        \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
      }
  }
```

Enf of \AddToHook.

```
\cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
  {
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
    \int_set:Nn \l_@@_initial_i_int { #1 }
    \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

For the column, it's a bit more complicated.

```
  \int_compare:nNnTF { #2 } = 1
    {
      \int_set:Nn \l_@@_initial_j_int 1
      \bool_set_true:N \l_@@_initial_open_bool
    }
    {
      \cs_if_exist:cTF
        {
          pgf @ sh @ ns @ \@@_env:
          - \int_use:N \l_@@_initial_i_int
          - \int_eval:n { #2 - 1 }
        }
        { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
        {
          \int_set:Nn \l_@@_initial_j_int { #2 }
          \bool_set_true:N \l_@@_initial_open_bool
        }
    }
  \int_compare:nNnTF { #2 + #3 -1 } = \c@jCol
    {
      \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
      \bool_set_true:N \l_@@_final_open_bool
    }
    {
      \cs_if_exist:cTF
        {
          pgf @ sh @ ns @ \@@_env:
          - \int_use:N \l_@@_final_i_int
          - \int_eval:n { #2 + #3 }
        }
        { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
        {
          \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
          \bool_set_true:N \l_@@_final_open_bool
        }
    }
  \group_begin:
  \int_compare:nNnTF { #1 } = 0
    { \color { nicematrix-first-row } }
    {
      \int_compare:nNnT { #1 } = \g_@@_row_total_int
        { \color { nicematrix-last-row } }
    }
  \keys_set:nn { NiceMatrix / xdots } { #4 }
  \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
  \@@_actually_draw_Ldots:
  \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).

```
  \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
    { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
 }
```

```
\hook_gput_code:nnn { begindocument } { . }
 {
  \tl_set:Nn \l_@@_argspec_tl { O { } m O { } E { _ ^ } { { } { } } }
  \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
  \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
    {
        \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
          {
            \@@_Vdotsfor:nnnn
              { \int_use:N \c@iRow }
              { \int_use:N \c@jCol }
              { #2 }
              {
                #1 , #3 ,
                down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
              }
          }
      }
  }
```

Enf of `\AddToHook`.

```
\cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
  {
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
```

For the column, it's easy.

```
    \int_set:Nn \l_@@_initial_j_int { #2 }
    \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

For the row, it's a bit more complicated.

```
    \int_compare:nNnTF #1 = 1
      {
        \int_set:Nn \l_@@_initial_i_int 1
        \bool_set_true:N \l_@@_initial_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_eval:n { #1 - 1 }
            - \int_use:N \l_@@_initial_j_int
          }
          { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
          {
            \int_set:Nn \l_@@_initial_i_int { #1 }
            \bool_set_true:N \l_@@_initial_open_bool
          }
      }
    \int_compare:nNnTF { #1 + #3 -1 } = \c@iRow
      {
        \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_eval:n { #1 + #3 }
            - \int_use:N \l_@@_final_j_int
          }
          { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
          {
            \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
            \bool_set_true:N \l_@@_final_open_bool
          }
      }

    \group_begin:
    \int_compare:nNnTF { #2 } = 0
      { \color { nicematrix-first-col } }
```

```
   {
    \int_compare:nNnT { #2 } = \g_@@_col_total_int
     { \color { nicematrix-last-col } }
   }
  \keys_set:nn { NiceMatrix / xdots } { #4 }
  \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
  \@@_actually_draw_Vdots:
  \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).

```
  \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
   { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
 }
```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```
\cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }
```

The command \line accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i*-*j*) and draws a dotted line between these cells.

First, we write a command with an argument of the format *i*-*j* and applies the command `\int_eval:n` to *i* and *j* ; this must *not* be protected (and is, of course fully expandable).[70]

```
\cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
 { \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).

```
\hook_gput_code:nnn { begindocument } { . }
 {
  \tl_set:Nn \l_@@_argspec_tl { O { } m m ! O { } E { _ ^ } { { } { } } }
  \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
  \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
   {
    \group_begin:
    \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
    \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
     \use:e
      {
       \@@_line_i:nn
        { \@@_double_int_eval:n #2 \q_stop }
        { \@@_double_int_eval:n #3 \q_stop }
      }
    \group_end:
   }
 }
```

[70] Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```
\cs_new_protected:Npn \@@_line_i:nn #1 #2
  {
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
    \bool_if:nTF
      {
        \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
          ||
        \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
      }
      {
        \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
      }
      { \@@_draw_line_ii:nn { #1 } { #2 } }
  }
\hook_gput_code:nnn { begindocument } { . }
  {
    \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
      {
```

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:`.

```
        \c_@@_pgfortikzpicture_tl
        \@@_draw_line_iii:nn { #1 } { #2 }
        \c_@@_endpgfortikzpicture_tl
      }
  }
```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```
\cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
  {
    \pgfrememberpicturepositiononpagetrue
    \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
    \@@_draw_line:
  }
```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

The command \RowStyle

```
\keys_define:nn { NiceMatrix / RowStyle }
  {
    cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
    cell-space-top-limit .initial:n = \c_zero_dim ,
    cell-space-top-limit .value_required:n = true ,
    cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
    cell-space-bottom-limit .initial:n = \c_zero_dim ,
    cell-space-bottom-limit .value_required:n = true ,
    cell-space-limits .meta:n =
      {
        cell-space-top-limit = #1 ,
        cell-space-bottom-limit = #1 ,
      } ,
    color .tl_set:N = \l_tmpa_tl ,
    color .value_required:n = true ,
    bold .bool_set:N = \l_tmpa_bool ,
    bold .default:n = true ,
    bold .initial:n = false ,
    nb-rows .int_set:N = \l_@@_key_nb_rows_int ,
    nb-rows .value_required:n = true ,
    nb-rows .initial:n = 1 ,
    rowcolor .tl_set:N = \l_@@_tmpc_tl ,
    rowcolor .value_required:n = true ,
    rowcolor .initial:n = ,
    unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
  }
```

```
\NewDocumentCommand \@@_RowStyle:n { O { } m }
  {
    \keys_set:nn { NiceMatrix / RowStyle } { #1 }
```

If the key `rowcolor` has been used.

```
    \tl_if_empty:NF \l_@@_tmpc_tl
      {
```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```
        \tl_gput_right:Nx \g_nicematrix_code_before_tl
          {
            \@@_rectanglecolor
              { \l_@@_tmpc_tl }
              { \int_use:N \c@iRow - \int_use:N \c@jCol }
              { \int_use:N \c@iRow - * }
          }
```

Then, the other rows (if there is several rows).

```
        \int_compare:nNnT \l_@@_key_nb_rows_int > 1
          {
            \tl_gput_right:Nx \g_nicematrix_code_before_tl
              {
                \@@_rowcolor
                  { \l_@@_tmpc_tl }
                  {
                    \int_eval:n { \c@iRow + 1 }
                    - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
                  }
              }
          }
      }
    \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
    \tl_gput_right:Nx \g_@@_row_style_tl
      { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
    \tl_gput_right:Nn \g_@@_row_style_tl { #2 }
```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```
    \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
      {
        \tl_gput_right:Nx \g_@@_row_style_tl
          {
            \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
              {
                \dim_set:Nn \l_@@_cell_space_top_limit_dim
                  { \dim_use:N \l_tmpa_dim }
              }
          }
      }
```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```
    \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
      {
        \tl_gput_right:Nx \g_@@_row_style_tl
```

```
          {
            \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
              {
                \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
                  { \dim_use:N \l_tmpb_dim }
              }
          }
      }
```

\l_tmpa_tl is the value of the key color of \RowStyle.

```
    \tl_if_empty:NF \l_tmpa_tl
      {
        \tl_gput_right:Nx \g_@@_row_style_tl
          { \mode_leave_vertical: \exp_not:N \color { \l_tmpa_tl } }
      }
```

\l_tmpa_bool is the value of the key bold.

```
    \bool_if:NT \l_tmpa_bool
      {
        \tl_gput_right:Nn \g_@@_row_style_tl
          {
            \if_mode_math:
              \c_math_toggle_token
              \bfseries \boldmath
              \c_math_toggle_token
            \else:
              \bfseries \boldmath
            \fi:
          }
      }
    \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
    \g_@@_row_style_tl
    \ignorespaces
  }
```

Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That's why we try to draw rectangles of the same color in the same instruction \pgfusepath { fill } (and they will be in the same instruction fill—coded f—in the resulting PDF).
The commands \@@_rowcolor, \@@_columncolor, \@@_rectanglecolor and \@@_rowlistcolors don't directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence \g_@@_colors_seq will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: [gray]{0.5}).
- For the color whose index in \g_@@_colors_seq is equal to i, a list of instructions which use that color will be constructed in the token list \g_@@_color_i_tl. In that token list, the instructions will be written using \@@_cartesian_color:nn and \@@_rectanglecolor:nn.

#1 is the color and #2 is an instruction using that color. Despite its name, the command \@@_add_to_colors_seq:nn doesn't only add a color to \g_@@_colors_seq: it also updates the corresponding token list \g_@@_color_i_tl. We add in a global way because the final user may use the instructions such as \cellcolor in a loop of pgffor in the \CodeBefore (and we recall that a loop of pgffor is encapsulated in a group).

```
\cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
  {
```

Firt, 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.

```
    \int_zero:N \l_tmpa_int
    \seq_map_indexed_inline:Nn \g_@@_colors_seq
      { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
    \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

First, the case where the color is a *new* color (not in the sequence).

```
      {
        \seq_gput_right:Nn \g_@@_colors_seq { #1 }
        \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
      }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
  }
```

```
\cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
\cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x x }
```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_`*i*`_tl`).

```
\cs_new_protected:Npn \@@_actually_color:
  {
    \pgfpicture
    \pgf@relevantforpicturesizefalse
    \seq_map_indexed_inline:Nn \g_@@_colors_seq
      {
        \color ##2
        \use:c { g_@@_color _ ##1 _tl }
        \tl_gclear:c { g_@@_color _ ##1 _tl }
        \pgfusepath { fill }
      }
    \endpgfpicture
  }
```

```
\cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
  {
    \tl_set:Nn \l_@@_rows_tl { #1 }
    \tl_set:Nn \l_@@_cols_tl { #2 }
    \@@_cartesian_path:
  }
```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
\NewDocumentCommand \@@_rowcolor { O { } m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \@@_add_to_colors_seq:xn
          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
          { \@@_cartesian_color:nn { #3 } { - } }
      }
  }
```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```
\NewDocumentCommand \@@_columncolor { O { } m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \@@_add_to_colors_seq:xn
          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
          { \@@_cartesian_color:nn { - } { #3 } }
    }
  }
```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
\NewDocumentCommand \@@_rectanglecolor { O { } m m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \@@_add_to_colors_seq:xn
          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
          { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
      }
  }
```

The last argument is the radius of the corners of the rectangle.

```
\NewDocumentCommand \@@_roundedrectanglecolor { O { } m m m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \@@_add_to_colors_seq:xn
          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
          { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
      }
  }
```

The last argument is the radius of the corners of the rectangle.

```
\cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
  {
    \@@_cut_on_hyphen:w #1 \q_stop
    \tl_clear_new:N \l_@@_tmpc_tl
    \tl_clear_new:N \l_@@_tmpd_tl
    \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
    \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
    \@@_cut_on_hyphen:w #2 \q_stop
    \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
    \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`.

```
    \@@_cartesian_path:n { #3 }
  }
```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```
\NewDocumentCommand \@@_cellcolor { O { } m m }
  {
    \clist_map_inline:nn { #3 }
      { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
  }

\NewDocumentCommand \@@_chessboardcolors { O { } m m  }
  {
    \int_step_inline:nn { \int_use:N \c@iRow }
      {
        \int_step_inline:nn { \int_use:N \c@jCol }
          {
            \int_if_even:nTF { ####1 + ##1 }
              { \@@_cellcolor [ #1 ] { #2 } }
              { \@@_cellcolor [ #1 ] { #3 } }
            { ##1 - ####1 }
          }
      }
  }
```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns) and the cells in the "corners".

```
\NewDocumentCommand \@@_arraycolor { O { } m }
  {
    \@@_rectanglecolor [ #1 ] { #2 }
      { 1 - 1 }
      { \int_use:N \c@iRow - \int_use:N \c@jCol }
  }
```

```
\keys_define:nn { NiceMatrix / rowcolors }
  {
    respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
    respect-blocks .default:n = true ,
    cols .tl_set:N = \l_@@_cols_tl ,
    restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
    restart .default:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
  }
```

The command `\rowcolors` (accessible in the `code-before`) is inspired by the command `\rowcolors` of the package xcolor (with the option `table`). However, the command `\rowcolors` of nicematrix has *not* the optional argument of the command `\rowcolors` of xcolor. Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.
`#1` (optional) is the color space ; `#2` is a list of intervals of rows ; `#3` is the list of colors ; `#4` is for the optional list of pairs *key=value*.

```
\NewDocumentCommand \@@_rowlistcolors { O { } m m O { } }
  {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```
    \group_begin:
    \seq_clear_new:N \l_@@_colors_seq
    \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
    \tl_clear_new:N \l_@@_cols_tl
    \tl_set:Nn \l_@@_cols_tl { - }
    \keys_set:nn { NiceMatrix / rowcolors } { #4 }
```

The counter `\l_@@_color_int` will be the rank of the current color in the list of colors (modulo the length of the list).

```
    \int_zero_new:N \l_@@_color_int
    \int_set:Nn \l_@@_color_int 1
    \bool_if:NT \l_@@_respect_blocks_bool
      {
```

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`).

```
        \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
        \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
          { \@@_not_in_exterior_p:nnnnn ##1 }
      }
    \pgfpicture
    \pgf@relevantforpicturesizefalse
```

`#2` is the list of intervals of rows.

```
    \clist_map_inline:nn { #2 }
      {
        \tl_set:Nn \l_tmpa_tl { ##1 }
        \tl_if_in:NnTF \l_tmpa_tl { - }
          { \@@_cut_on_hyphen:w ##1 \q_stop }
          { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
```

Now, l_tmpa_tl and l_tmpb_tl are the first row and the last row of the interval of rows that we have to treat. The counter \l_tmpa_int will be the index of the loop over the rows.

```
        \int_set:Nn \l_tmpa_int \l_tmpa_tl
        \bool_if:NTF \l_@@_rowcolors_restart_bool
          { \int_set:Nn \l_@@_color_int 1 }
          { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
        \int_zero_new:N \l_@@_tmpc_int
        \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
        \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
          {
```

We will compute in \l_tmpb_int the last row of the “block”.

```
            \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).

```
            \bool_if:NT \l_@@_respect_blocks_bool
              {
                \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
                  { \@@_intersect_our_row_p:nnnnn ####1 }
                \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }
```

Now, the last row of the block is computed in \l_tmpb_int.

```
              }
            \tl_set:Nx \l_@@_rows_tl
              { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

\l_@@_tmpc_tl will be the color that we will use.

```
            \tl_clear_new:N \l_@@_color_tl
            \tl_set:Nx \l_@@_color_tl
              {
                \@@_color_index:n
                  {
                    \int_mod:nn
                      { \l_@@_color_int - 1 }
                      { \seq_count:N \l_@@_colors_seq }
                    + 1
                  }
              }
            \tl_if_empty:NF \l_@@_color_tl
              {
                \@@_add_to_colors_seq:xx
                  { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
                  { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
              }
            \int_incr:N \l_@@_color_int
            \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
          }
      }
    \endpgfpicture
    \group_end:
  }
```

The command \@@_color_index:n peeks in \l_@@_colors_seq the color at the index #1. However, if that color is the symbol =, the previous one is poken. This macro is recursive.

```
\cs_new:Npn \@@_color_index:n #1
  {
    \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
      { \@@_color_index:n { #1 - 1 } }
      { \seq_item:Nn \l_@@_colors_seq { #1 } }
  }
```

The command \rowcolors (available in the \CodeBefore) is a specialisation of the most general command \rowlistcolors.

```
\NewDocumentCommand \@@_rowcolors { O { } m m m O { } }
  { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }
```

```
\cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
  {
    \int_compare:nNnT { #3 } > \l_tmpb_int
      { \int_set:Nn \l_tmpb_int { #3 } }
  }
```

```
\prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
  {
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn { #4 } = \c_zero_int }
      { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
      \prg_return_false:
      \prg_return_true:
  }
```

The following command return `true` when the block intersects the row `\l_tmpa_int`.

```
\prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
  {
    \bool_if:nTF
      {
        \int_compare_p:n { #1 <= \l_tmpa_int }
        &&
        \int_compare_p:n { \l_tmpa_int <= #3 }
      }
      \prg_return_true:
      \prg_return_false:
  }
```

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`).

```
\cs_new_protected:Npn \@@_cartesian_path:n #1
  {
    \bool_lazy_and:nnT
      { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
      { \dim_compare_p:nNn { #1 } = \c_zero_dim }
      {
        \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
        \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
      }
```

We begin the loop over the columns.

```
    \clist_map_inline:Nn \l_@@_cols_tl
      {
        \tl_set:Nn \l_tmpa_tl { ##1 }
        \tl_if_in:NnTF \l_tmpa_tl { - }
          { \@@_cut_on_hyphen:w ##1 \q_stop }
          { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
        \bool_lazy_or:nnT
          { \tl_if_blank_p:V \l_tmpa_tl }
          { \str_if_eq_p:Vn \l_tmpa_tl { * } }
          { \tl_set:Nn \l_tmpa_tl { 1 } }
        \bool_lazy_or:nnT
          { \tl_if_blank_p:V \l_tmpb_tl }
          { \str_if_eq_p:Vn \l_tmpb_tl { * } }
          { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
        \int_compare:nNnT \l_tmpb_tl > \c@jCol
          { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
```

\l_@@_tmpc_tl will contain the number of column.

```
        \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.

```
        \@@_qpoint:n { col - \l_tmpa_tl }
        \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
          { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
          { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
        \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 }  }
        \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
        \clist_map_inline:Nn \l_@@_rows_tl
          {
            \tl_set:Nn \l_tmpa_tl { ####1 }
            \tl_if_in:NnTF \l_tmpa_tl { - }
              { \@@_cut_on_hyphen:w ####1 \q_stop }
              { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
            \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
            \tl_if_empty:NT \l_tmpb_tl
              { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
            \int_compare:nNnT \l_tmpb_tl > \c@iRow
              { \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.

```
            \seq_if_in:NxF \l_@@_corners_cells_seq
              { \l_tmpa_tl - \l_@@_tmpc_tl }
              {
                \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
                \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
                \@@_qpoint:n { row - \l_tmpa_tl }
                \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
                \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
                \pgfpathrectanglecorners
                  { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
                  { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
              }
          }
      }
  }
```

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).

```
\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.

```
\cs_new_protected:Npn \@@_expand_clist:NN #1 #2
  {
    \clist_set_eq:NN \l_tmpa_clist #1
    \clist_clear:N #1
    \clist_map_inline:Nn \l_tmpa_clist
      {
        \tl_set:Nn \l_tmpa_tl { ##1 }
        \tl_if_in:NnTF \l_tmpa_tl { - }
          { \@@_cut_on_hyphen:w ##1 \q_stop }
          { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
        \bool_lazy_or:nnT
          { \tl_if_blank_p:V \l_tmpa_tl }
          { \str_if_eq_p:Vn \l_tmpa_tl { * } }
```

```
            { \tl_set:Nn \l_tmpa_tl { 1 } }
          \bool_lazy_or:nnT
            { \tl_if_blank_p:V \l_tmpb_tl }
            { \str_if_eq_p:Vn \l_tmpb_tl { * } }
            { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
          \int_compare:nNnT \l_tmpb_tl > #2
            { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
          \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
            { \clist_put_right:Nn #1 { ####1 } }
        }
    }
```

When the user uses the key `colortbl-like`, the following command will be linked to `\cellcolor` in the tabular.

```
\NewDocumentCommand \@@_cellcolor_tabular { O { } m }
  {
    \peek_remove_spaces:n
      {
        \tl_gput_right:Nx \g_nicematrix_code_before_tl
          {
```

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).

```
            \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
              { \int_use:N \c@iRow - \int_use:N \c@jCol }
          }
      }
  }
```

When the user uses the key `colortbl-like`, the following command will be linked to `\rowcolor` in the tabular.

```
\NewDocumentCommand \@@_rowcolor_tabular { O { } m }
  {
    \peek_remove_spaces:n
      {
        \tl_gput_right:Nx \g_nicematrix_code_before_tl
          {
            \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
              { \int_use:N \c@iRow - \int_use:N \c@jCol }
              { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
          }
      }
  }
```

```
\NewDocumentCommand \@@_columncolor_preamble { O { } m }
  {
```

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).

```
    \int_compare:nNnT \c@jCol > \g_@@_col_total_int
      {
```

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).

```
        \tl_gput_left:Nx \g_nicematrix_code_before_tl
          {
            \exp_not:N \columncolor [ #1 ]
              { \exp_not:n { #2 } } { \int_use:N \c@jCol }
          }
      }
  }
```

The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumntype` 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`.

```
\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`.

```
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
  {
    \int_compare:nNnTF \l_@@_first_col_int = 0
      { \@@_OnlyMainNiceMatrix_i:n { #1 } }
      {
        \int_compare:nNnTF \c@jCol = 0
          {
            \int_compare:nNnF \c@iRow = { -1 }
              { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
          }
          { \@@_OnlyMainNiceMatrix_i:n { #1 } }
      }
  }
```

This definition may seem complicated but we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell.

The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
  {
    \int_compare:nNnF \c@iRow = 0
      { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
  }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or "subsystem" of nicematrix wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```
\keys_define:nn { NiceMatrix / Rules }
  {
    position .int_set:N = \l_@@_position_int ,
    position .value_required:n = true ,
    start .int_set:N = \l_@@_start_int ,
    start .initial:n = 1 ,
    end .int_set:N = \l_@@_end_int ,
```

The following keys are no-op because there are keys which may be inherited from a list of pairs *key=value* of a definition of a customized rule (with the key `custom-line` of `\NiceMatrixOptions`).

```
    % letter .code:n = \prg_do_nothing: ,
    % command .code:n = \prg_do_nothing:
  }
```

It's possible that the rule won't be drawn continuously from `start` ot `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```
\keys_define:nn { NiceMatrix / RulesBis }
  {
    multiplicity .int_set:N = \l_@@_multiplicity_int ,
    multiplicity .initial:n = 1 ,
    dotted .bool_set:N = \l_@@_dotted_bool ,
    dotted .initial:n = false ,
    dotted .default:n = true ,
    color .code:n = \@@_set_CT@arc@: #1 \q_stop ,
    color .value_required:n = true ,
    sep-color .code:n = \@@_set_CT@drsc@: #1 \q_stop ,
    sep-color .value_required:n = true ,
```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with Tikz.

```
    tikz .tl_set:N = \l_@@_tikz_rule_tl ,
    tikz .value_required:n = true ,
    tikz .initial:n = ,
    width .dim_set:N = \l_@@_rule_width_dim ,
    width .value_required:n = true
  }
```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of *key=value* pairs.

```
\cs_new_protected:Npn \@@_vline:n #1
  {
```

The group is for the options.

```
    \group_begin:
    \int_zero_new:N \l_@@_end_int
    \int_set_eq:NN \l_@@_end_int \c@iRow
    \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```
    \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
      \@@_vline_i:
    \group_end:
  }
\cs_new_protected:Npn \@@_vline_i:
  {
    \int_zero_new:N \l_@@_local_start_int
    \int_zero_new:N \l_@@_local_end_int
```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```
    \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
    \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
      \l_tmpa_tl
      {
```

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.

```
        \bool_gset_true:N \g_tmpa_bool
        \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
```

```
          { \@@_test_vline_in_block:nnnnn ##1 }
        \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
          { \@@_test_vline_in_block:nnnnn ##1 }
        \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
          { \@@_test_vline_in_stroken_block:nnnn ##1 }
        \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
        \bool_if:NTF \g_tmpa_bool
          {
            \int_compare:nNnT \l_@@_local_start_int = 0
```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```
              { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
          }
          {
            \int_compare:nNnT \l_@@_local_start_int > 0
              {
                \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
                \@@_vline_ii:
                \int_zero:N \l_@@_local_start_int
              }
          }
      }
    \int_compare:nNnT \l_@@_local_start_int > 0
      {
        \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
        \@@_vline_ii:
      }
  }

\cs_new_protected:Npn \@@_test_in_corner_v:
   {
     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
       {
         \seq_if_in:NxT
           \l_@@_corners_cells_seq
           { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
           { \bool_set_false:N \g_tmpa_bool }
       }
       {
         \seq_if_in:NxT
           \l_@@_corners_cells_seq
           { \l_tmpa_tl - \l_tmpb_tl }
           {
             \int_compare:nNnTF \l_tmpb_tl = 1
               { \bool_set_false:N \g_tmpa_bool }
               {
                 \seq_if_in:NxT
                   \l_@@_corners_cells_seq
                   { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
                   { \bool_set_false:N \g_tmpa_bool }
               }
           }
       }
   }

\cs_new_protected:Npn \@@_vline_ii:
  {
    \bool_set_false:N \l_@@_dotted_boo
```

We use `\keys_set_known:nV` and not `\keys_set:nV` because there may be the keys `letter` and `command` in the list (these keys are present if the rule comes from a customized line (created by `custom-line`).

```
    \keys_set_known:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
    \bool_if:NTF \l_@@_dotted_bool
      \@@_vline_iv:
      {
        \tl_if_empty:NTF \l_@@_tikz_rule_tl
          \@@_vline_iii:
          \@@_vline_v:
      }
  }
```

First the case of a standard rule, that is to say a rule which is not dotted (and the user has not used the key tikz).

```
\cs_new_protected:Npn \@@_vline_iii:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
    \dim_set_eq:NN \l_tmpb_dim \pgf@x
    \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
    \bool_lazy_all:nT
      {
        { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
        { \cs_if_exist_p:N \CT@drsc@ }
        { ! \tl_if_blank_p:V \CT@drsc@ }
      }
      {
        \group_begin:
        \CT@drsc@
        \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
        \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
        \dim_set:Nn \l_@@_tmpd_dim
          {
            \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
            * ( \l_@@_multiplicity_int - 1 )
          }
        \pgfpathrectanglecorners
          { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
          { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
        \pgfusepath { fill }
        \group_end:
      }
    \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
    \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
    \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
      {
        \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
        \dim_sub:Nn \l_tmpb_dim \doublerulesep
        \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
        \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
      }
    \CT@arc@
    \pgfsetlinewidth { 1.1 \arrayrulewidth }
    \pgfsetrectcap
    \pgfusepathqstroke
    \endpgfpicture
  }
```

The following code is for the case of a dotted rule (with our system of rounded dots).

```
\cs_new_protected:Npn \@@_vline_iv:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
    \CT@arc@
    \@@_draw_line:
    \endpgfpicture
  }
```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```
\cs_new_protected:Npn \@@_vline_v:
  {
    \begin {tikzpicture }
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
    \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
    \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
    \exp_args:NV \tikzset \l_@@_tikz_rule_tl
    \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
      ( \l_tmpb_dim , \l_tmpa_dim ) --
      ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
    \end { tikzpicture }
  }
```

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).

```
\cs_new_protected:Npn \@@_draw_vlines:
  {
    \int_step_inline:nnn
      {
        \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
          1 2
      }
      {
        \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
          { \int_eval:n { \c@jCol + 1 } }
          \c@jCol
      }
      {
        \tl_if_eq:NnF \l_@@_vlines_clist { all }
          { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
          { \@@_vline:n { position = ##1 } }
      }
  }
```

The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of *key=value* pairs of the form `{NiceMatrix/Rules}`.

```
\cs_new_protected:Npn \@@_hline:n #1
  {
```

The group is for the options.

```
    \group_begin:
    \int_zero_new:N \l_@@_end_int
    \int_set_eq:NN \l_@@_end_int \c@jCol
    \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
    \@@_hline_i:
    \group_end:
  }
```

```
\cs_new_protected:Npn \@@_hline_i:
  {
    \int_zero_new:N \l_@@_local_start_int
    \int_zero_new:N \l_@@_local_end_int
```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a column corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```
    \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
    \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
      \l_tmpb_tl
      {
```

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.

```
         \bool_gset_true:N \g_tmpa_bool
         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
           { \@@_test_hline_in_block:nnnnn ##1 }
         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
           { \@@_test_hline_in_block:nnnnn ##1 }
         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
           { \@@_test_hline_in_stroken_block:nnnn ##1 }
         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
         \bool_if:NTF \g_tmpa_bool
           {
             \int_compare:nNnT \l_@@_local_start_int = 0
```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```
               { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
           }
           {
             \int_compare:nNnT \l_@@_local_start_int > 0
               {
                 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
                 \@@_hline_ii:
                 \int_zero:N \l_@@_local_start_int
               }
           }
      }
    \int_compare:nNnT \l_@@_local_start_int > 0
      {
        \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
        \@@_hline_ii:
      }
  }
```

```
\cs_new_protected:Npn \@@_test_in_corner_h:
   {
     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
       {
         \seq_if_in:NxT
           \l_@@_corners_cells_seq
           { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
           { \bool_set_false:N \g_tmpa_bool }
       }
       {
         \seq_if_in:NxT
           \l_@@_corners_cells_seq
           { \l_tmpa_tl - \l_tmpb_tl }
           {
             \int_compare:nNnTF \l_tmpa_tl = 1
               { \bool_set_false:N \g_tmpa_bool }
               {
                 \seq_if_in:NxT
                   \l_@@_corners_cells_seq
                   { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
                   { \bool_set_false:N \g_tmpa_bool }
               }
           }
       }
   }
```

```
\cs_new_protected:Npn \@@_hline_ii:
  {
    \bool_set_false:N \l_@@_dotted_bool
```

We use `\keys_set_known:nV` and not `\keys_set:nV` because there may be the keys `letter` and `command` in the list (these keys are present if the rule comes from a customized line (created by `custom-line`).

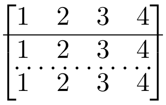
```
    \keys_set_known:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
    \bool_if:NTF \l_@@_dotted_bool
      \@@_hline_iv:
      {
        \tl_if_empty:NTF \l_@@_tikz_rule_tl
          \@@_hline_iii:
          \@@_hline_v:
      }
  }
```

First the case of a standard rule, that is to say a rule which is not dotted.

```
\cs_new_protected:Npn \@@_hline_iii:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_tmpa_dim \pgf@x
    \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
    \dim_set_eq:NN \l_tmpb_dim \pgf@y
    \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
    \bool_lazy_all:nT
      {
        { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
        { \cs_if_exist_p:N \CT@drsc@ }
        { ! \tl_if_blank_p:V \CT@drsc@ }
      }
      {
        \group_begin:
        \CT@drsc@
        \dim_set:Nn \l_@@_tmpd_dim
          {
            \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
            * ( \l_@@_multiplicity_int - 1 )
          }
        \pgfpathrectanglecorners
          { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
          { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
        \pgfusepathqfill
        \group_end:
      }
    \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
    \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
    \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
      {
        \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
        \dim_sub:Nn \l_tmpb_dim \doublerulesep
        \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
        \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
      }
    \CT@arc@
    \pgfsetlinewidth { 1.1 \arrayrulewidth }
    \pgfsetrectcap
    \pgfusepathqstroke
    \endpgfpicture
  }
```

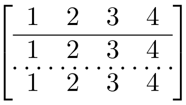
The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```



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}
```



```
\cs_new_protected:Npn \@@_hline_iv:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
    \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \int_compare:nNnT \l_@@_local_start_int = 1
      {
        \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
        \bool_if:NT \l_@@_NiceArray_bool
          { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }
```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by `0.5 \l_@@_inter_dots_dim` is *ad hoc* for a better result.

```
        \tl_if_eq:NnF \g_@@_left_delim_tl (
          { \dim_add:Nn \l_@@_x_initial_dim  { 0.5 \l_@@_inter_dots_dim } }
```

```
      }
    \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \int_compare:nNnT \l_@@_local_end_int = \c@jCol
      {
        \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
        \bool_if:NT \l_@@_NiceArray_bool
          { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
        \tl_if_eq:NnF \g_@@_right_delim_tl )
          { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
      }
    \CT@arc@
    \@@_draw_line:
    \endpgfpicture
  }
```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```
\cs_new_protected:Npn \@@_hline_v:
  {
    \begin { tikzpicture }
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_tmpa_dim \pgf@x
    \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
    \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
    \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
    \exp_args:NV \tikzset \l_@@_tikz_rule_tl
    \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
      ( \l_tmpa_dim , \l_tmpb_dim ) --
      ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
    \end { tikzpicture }
  }
```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners (if the key `corners` is used)).

```
\cs_new_protected:Npn \@@_draw_hlines:
  {
    \int_step_inline:nnn
      {
        \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
          1 2
      }
      {
        \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
          { \int_eval:n { \c@iRow + 1 } }
          \c@iRow
      }
      {
        \tl_if_eq:NnF \l_@@_hlines_clist { all }
          { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
          { \@@_hline:n { position = ##1 } }
      }
  }
```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of nicematrix.

```
\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.

```
\cs_set:Npn \@@_Hline_i:n #1
  {
    \peek_meaning_ignore_spaces:NTF \Hline
      { \@@_Hline_ii:nn { #1 + 1 } }
      { \@@_Hline_iii:n { #1 } }
  }
```

```
\cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
```

```
\cs_set:Npn \@@_Hline_iii:n #1
  {
    \skip_vertical:n
      {
        \arrayrulewidth * ( #1 )
        + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
      }
    \tl_gput_right:Nx \g_@@_internal_code_after_tl
      {
        \@@_hline:n
          {
            position = \int_eval:n { \c@iRow + 1 } ,
            multiplicity = #1
          }
      }
    \ifnum 0 = `{ \fi }
  }
```

Customized rules defined by the final user

The final user can define a customized rule by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs.

Among the keys avalaible in that list, there is the key `letter` to specify a letter that the final user will use in the preamble of the array. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix / ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```
\keys_define:nn { NiceMatrix / ColumnTypes }  { }
```

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```
\cs_new_protected:Npn \@@_custom_line:n #1
  {
    \str_clear_new:N \l_@@_command_str
    \str_clear_new:N \l_@@_letter_str
    \dim_zero_new:N \l_@@_rule_width_dim
```

The token list `\l_tmpa_tl` is for the key `color`.

```
    \tl_clear:N \l_tmpa_tl
```

The flag `\l_tmpa_bool` will indicate whether the key `tikz` is present.

```
    \bool_set_false:N \l_tmpa_bool
```

The flag `\l_tmpb_bool` will indicate whether the key `width` is present.

```
    \bool_set_false:N \l_tmpb_bool
    \keys_set_known:nn { NiceMatrix / Custom-Line } { #1 }
    \bool_if:NT \l_tmpa_bool
      {
```

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.

```
        \cs_if_exist:NF \tikzpicture
          { \@@_error:n { tikz~in~custom-line~without~tikz } }
        \tl_if_empty:NF \l_tmpa_tl
```

```
     { \@@_error:n { color~in~custom-line~with~tikz } }
   }
  \bool_if:NT \l_tmpb_bool
   {
    \bool_if:NF \l_tmpa_bool
     { \@@_error:n { key~width~without~key~tikz } }
   }
```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```
  \bool_lazy_and:nnTF
   { \str_if_empty_p:N \l_@@_letter_str }
   { \str_if_empty_p:N \l_@@_command_str }
   { \@@_error:n { No~letter~and~no~command } }
   {
    \str_if_empty:NF \l_@@_letter_str
     {
      \int_compare:nNnTF { \str_count:N \l_@@_letter_str } = 1
       {
        \exp_args:NnV \tl_if_in:NnTF
         \c_@@_forbidden_letters_str \l_@@_letter_str
         { \@@_error:n { Forbidden~letter } }
         {
```

The final user can, locally, redefine a letter of column type. That's compatible with the use of `\keys_define:nn`: the definition is local and may overwrite a previous definition.

```
          \keys_define:nx { NiceMatrix / ColumnTypes }
           {
            \l_@@_letter_str .code:n =
             { \@@_custom_line_i:n { \exp_not:n { #1 } } }
           }
         }
       }
       { \@@_error:n { Several~letters } }
     }
    \str_if_empty:NF \l_@@_command_str
     {
```

The flag `\l_tmpa_bool` means that the key 'tikz' have been used. When the key 'tikz' has not been used, the width of the rule is computed with the multiplicity of the rule.

```
      \bool_if:NF \l_tmpa_bool
       {
        \dim_set:Nn \l_@@_rule_width_dim
         {
          \arrayrulewidth * \l_@@_tmpc_int
          + \doublerulesep * ( \l_@@_tmpc_int - 1 )
         }
       }
      \@@_define_h_custom_line:nV { #1 } \l_@@_rule_width_dim
     }
   }
 }
\str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }
```

The previous command `\@@_custom_line:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has only entries for a few keys.

```
\keys_define:nn { NiceMatrix / Custom-Line }
 {
  % here, we will use change in the future to use .tl_set:N
```

```
    letter .code:n = \str_set:Nn \l_@@_letter_str { #1 } ,
    letter .value_required:n = true ,
    % here, we will use change in the future to use .tl_set:N
    command .code:n = \str_set:Nn \l_@@_command_str { #1 } ,
    command .value_required:n = true ,
    multiplicity .int_set:N = \l_@@_tmpc_int ,
    multiplicity .initial:n = 1 ,
    multiplicity .value_required:n = true ,
    color .tl_set:N = \l_tmpa_tl ,
    color .value_required:n = true ,
```

When the key tikz is used, the rule will be drawn with Tikz by using the set of keys specified in the value of that key tikz.

```
    tikz .code:n = \bool_set_true:N \l_tmpa_bool ,
```

The key width must be used only when the key tikz is used. When used, the key width specifies the width of the rule: it will be used to reserve space (in the preamble of the array or in the command for the horizontal rules).

```
    width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
                    \bool_set_true:N \l_tmpb_bool ,
    width .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
  }
```

The following command will create the command that the final user will use in its array to draw a horizontal rule (hence the 'h' in the name). #1 is the whole set of keys to pass to \@@_line:n and #2 is the widht of the whole rule.

```
\cs_new_protected:Npn \@@_define_h_custom_line:nn #1 #2
  {
```

We use \cs_set:cpn and not \cs_new:cpn because we want a local definition. Moreover, the command must *not* be protected since it begins with \noalign.

```
    \cs_set:cpn { nicematrix - \l_@@_command_str }
      {
        \noalign
          {
            \skip_vertical:n { #2 }
            \tl_gput_right:Nx \g_@@_internal_code_after_tl
              { \@@_hline:n { #1 , position = \int_eval:n { \c@iRow + 1 } } }
          }
      }
    \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
  }
\cs_generate_variant:Nn \@@_define_h_custom_line:nn { n V }
```

The flag \l_tmpa_bool means that the key 'tikz' have been used. When the key 'tikz' has not been used, the width of the rule is computed with the multiplicity of the rule.

```
\cs_new_protected:Npn \@@_custom_line_i:n #1
  {
    \bool_if:NF \l_tmpa_bool
      {
        \dim_set:Nn \l_@@_rule_width_dim
          {
            \arrayrulewidth * \l_@@_tmpc_int
            + \doublerulesep * ( \l_@@_tmpc_int - 1)
          }
      }
    \tl_gput_right:Nx \g_@@_preamble_tl
      {
        \exp_not:N !
          { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } }
      }
     \tl_gput_right:Nx \g_@@_internal_code_after_tl
      { \@@_vline:n { #1 , position = \int_eval:n { \c@jCol + 1 } } }
  }
```

```
\@@_custom_line:n { letter = : , command = hdottedline , dotted }
```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments `#1`, `#2`, `#3` and `#4`. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to `false`.

```
\cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4
  {
    \bool_lazy_all:nT
      {
        { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
        { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
        { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
        { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
      }
      { \bool_gset_false:N \g_tmpa_bool }
  }
```

The same for vertical rules.

```
\cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4
  {
    \bool_lazy_all:nT
      {
        { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
        { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
        { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
        { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
      }
      { \bool_gset_false:N \g_tmpa_bool }
  }
\cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
  {
    \bool_lazy_all:nT
      {
        { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
        { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
        { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
        { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
      }
      { \bool_gset_false:N \g_tmpa_bool }
  }
\cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
  {
    \bool_lazy_all:nT
      {
        { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
        { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
        { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
        { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
      }
      { \bool_gset_false:N \g_tmpa_bool }
  }
```

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.

```
\cs_new_protected:Npn \@@_compute_corners:
  {
```

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.

```
    \seq_clear_new:N \l_@@_corners_cells_seq
    \clist_map_inline:Nn \l_@@_corners_clist
      {
        \str_case:nnF { ##1 }
          {
            { NW }
            { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
            { NE }
            { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
            { SW }
            { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
            { SE }
            { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
          }
          { \@@_error:nn { bad~corner } { ##1 } }
      }
```

Even if the user has used the key corners the list of cells in the corners may be empty.

```
    \seq_if_empty:NF \l_@@_corners_cells_seq
      {
```

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.

```
        \tl_gput_right:Nx \g_@@_aux_tl
          {
            \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
              { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
          }
      }
  }
```

“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.

```
\cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
  {
```

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.

```
    \bool_set_false:N \l_tmpa_bool
    \int_zero_new:N \l_@@_last_empty_row_int
    \int_set:Nn \l_@@_last_empty_row_int { #1 }
    \int_step_inline:nnnn { #1 } { #3 } { #5 }
      {
        \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
        \bool_lazy_or:nnTF
          {
            \cs_if_exist_p:c
              { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
```

```
     }
     \l_tmpb_bool
     { \bool_set_true:N \l_tmpa_bool }
     {
      \bool_if:NF \l_tmpa_bool
       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
     }
   }
```

Now, you determine the last empty cell in the row of number 1.

```
  \bool_set_false:N \l_tmpa_bool
  \int_zero_new:N \l_@@_last_empty_column_int
  \int_set:Nn \l_@@_last_empty_column_int { #2 }
  \int_step_inline:nnnn { #2 } { #4 } { #6 }
   {
    \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
    \bool_lazy_or:nnTF
     \l_tmpb_bool
     {
      \cs_if_exist_p:c
       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
     }
     { \bool_set_true:N \l_tmpa_bool }
     {
      \bool_if:NF \l_tmpa_bool
       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
     }
   }
```

Now, we loop over the rows.

```
  \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
   {
```

We treat the row number ##1 with another loop.

```
    \bool_set_false:N \l_tmpa_bool
    \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
     {
      \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
      \bool_lazy_or:nnTF
       \l_tmpb_bool
       {
        \cs_if_exist_p:c
         { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
       }
       { \bool_set_true:N \l_tmpa_bool }
       {
        \bool_if:NF \l_tmpa_bool
         {
          \int_set:Nn \l_@@_last_empty_column_int { ####1 }
          \seq_put_right:Nn
           \l_@@_corners_cells_seq
           { ##1 - ####1 }
         }
       }
     }
   }
 }
```

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).

```
\cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
 {
  \int_set:Nn \l_tmpa_int { #1 }
```

```
    \int_set:Nn \l_tmpb_int { #2 }
    \bool_set_false:N \l_tmpb_bool
    \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
      { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
  }
\cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
  {
    \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
      {
        \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
          {
            \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
              {
                \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
                  { \bool_set_true:N \l_tmpb_bool }
              }
          }
      }
  }
```

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`.

```
\cs_new:Npn \@@_hdottedline:
  {
    \noalign { \skip_vertical:N 2\l_@@_radius_dim }
    \@@_hdottedline_i:
  }
```

On the other side, the following command should be protected.

```
\cs_new_protected:Npn \@@_hdottedline_i:
  {
```

We write in the internal `\CodeAfter` the instruction that will actually draw the dotted line. It's not possible to draw this dotted line now because we don't know the length of the line (we don't even know the number of columns).

```
    \tl_gput_right:Nx \g_@@_internal_code_after_tl
      { \@@_hdottedline:n { \int_use:N \c@iRow } }
  }
```

The command `\@@_hdottedline:n` is the command written in the internal `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```
\cs_new_protected:Npn \@@_hdottedline:n #1
  { \@@_hline:n { position = #1 , end = \int_use:N \c@jCol , dotted } }
```

Vertical dotted lines

```
\cs_new_protected:Npn \@@_vdottedline:n #1
  { \@@_vline:n { position = \int_eval:n { #1 + 1 } , dotted } }
```

The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```
\bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
\keys_define:nn { NiceMatrix / NiceMatrixBlock }
  {
    auto-columns-width .code:n =
      {
        \bool_set_true:N \l_@@_block_auto_columns_width_bool
        \dim_gzero_new:N \g_@@_max_cell_width_dim
        \bool_set_true:N \l_@@_auto_columns_width_bool
      }
  }
```

```
\NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
  {
    \int_gincr:N \g_@@_NiceMatrixBlock_int
    \dim_zero:N \l_@@_columns_width_dim
    \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
    \bool_if:NT \l_@@_block_auto_columns_width_bool
      {
        \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
          {
            \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
              { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
          }
      }
  }
```

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).

```
  {
    \bool_if:NT \l_@@_block_auto_columns_width_bool
      {
        \iow_shipout:Nn \@mainaux \ExplSyntaxOn
        \iow_shipout:Nx \@mainaux
          {
            \cs_gset:cpn
              { @@ _ 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.

```
              { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
          }
        \iow_shipout:Nn \@mainaux \ExplSyntaxOff
      }
  }
```

The extra nodes

First, two variants of the functions \dim_min:nn and \dim_max:nn.

```
\cs_generate_variant:Nn \dim_min:nn { v n }
\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).

```
\cs_new_protected:Npn \@@_create_extra_nodes:
  {
    \bool_if:nTF \l_@@_medium_nodes_bool
      {
        \bool_if:NTF \l_@@_large_nodes_bool
          \@@_create_medium_and_large_nodes:
          \@@_create_medium_nodes:
      }
      { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
  }
```

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`.

```
\cs_new_protected:Npn \@@_computations_for_medium_nodes:
  {
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
        \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
        \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
        \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
      }
    \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
      {
        \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
        \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
        \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
        \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
      }
```

We begin the two nested loops over the rows and the columns of the array.

```
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \int_step_variable:nnNn
          \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.

```
          {
            \cs_if_exist:cT
              { 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`.

```
              {
                \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
                \dim_set:cn { l_@@_row_\@@_i: _min_dim}
```

```
                    { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
                  \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
                    {
                      \dim_set:cn { l_@@_column _ \@@_j: _min_dim}
                        { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
                    }
```

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`.

```
                  \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
                  \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
                    { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
                  \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
                    {
                      \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
                        { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
                    }
                }
            }
        }
```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \dim_compare:nNnT
          { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
          {
            \@@_qpoint:n { row - \@@_i: - base }
            \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
            \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
          }
      }
    \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
      {
        \dim_compare:nNnT
          { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
          {
            \@@_qpoint:n { col - \@@_j: }
            \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
            \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
          }
      }
  }
```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the "medium nodes" are created.

```
\cs_new_protected:Npn \@@_create_medium_nodes:
  {
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_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".

```
      \tl_set:Nn \l_@@_suffix_tl { -medium }
      \@@_create_nodes:
      \endpgfpicture
  }
```

The command \@@_create_large_nodes: must be used when we want to create only the "large nodes" and not the medium ones[71]. 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:.

```
\cs_new_protected:Npn \@@_create_large_nodes:
  {
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_computations_for_medium_nodes:
      \@@_computations_for_large_nodes:
      \tl_set:Nn \l_@@_suffix_tl { - large }
      \@@_create_nodes:
    \endpgfpicture
  }
```

```
\cs_new_protected:Npn \@@_create_medium_and_large_nodes:
  {
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_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".

```
      \tl_set:Nn \l_@@_suffix_tl { - medium }
      \@@_create_nodes:
      \@@_computations_for_large_nodes:
      \tl_set:Nn \l_@@_suffix_tl { - large }
      \@@_create_nodes:
    \endpgfpicture
  }
```

For "large nodes", the exterior rows and columns don't interfer. 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.

```
\cs_new_protected:Npn \@@_computations_for_large_nodes:
  {
    \int_set:Nn \l_@@_first_row_int 1
    \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.

```
    \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
      {
        \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
          {
            (
              \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
              \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 }  _ max _ dim }
            )
            / 2
          }
        \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
          { l_@@_row_\@@_i: _min_dim }
      }
    \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
      {
        \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
          {
```

[71] If we want to create both, we have to use \@@_create_medium_and_large_nodes:

```
            (
              \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
              \dim_use:c
                { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
            )
            / 2
          }
        \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
          { l_@@_column _ \@@_j: _ max _ dim }
      }
```

Here, we have to use \dim_sub:cn because of the number 1 in the name.

```
    \dim_sub:cn
      { l_@@_column _ 1 _ min _ dim }
      \l_@@_left_margin_dim
    \dim_add:cn
      { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
      \l_@@_right_margin_dim
  }
```

The command \@@_create_nodes: is used twice: for the construction of the “medium nodes” and for the construction of the “large nodes”. The nodes are constructed with the value of all the dimensions l_@@_row_*i*_min_dim, l_@@_row_*i*_max_dim, l_@@_column_*j*_min_dim and l_@@_column_*j*_max_dim. Between the construction of the “medium nodes” and the “large nodes”, the values of these dimensions are changed.
The function also uses \l_@@_suffix_tl (-medium or -large).

```
\cs_new_protected:Npn \@@_create_nodes:
  {
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
          {
```

We draw the rectangular node for the cell (\@@_i-\@@_j).

```
            \@@_pgf_rect_node:nnnnn
              { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
              { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
              { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
              { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
              { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
            \str_if_empty:NF \l_@@_name_str
              {
                \pgfnodealias
                  { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
                  { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
              }
          }
      }
```

Now, we create the nodes for the cells of the \multicolumn. We recall that we have stored in \g_@@_multicolumn_cells_seq the list of the cells where a \multicolumn{n}{...}{...} with n>1 was issued and in \g_@@_multicolumn_sizes_seq the correspondant values of n.

```
    \seq_mapthread_function:NNN
      \g_@@_multicolumn_cells_seq
      \g_@@_multicolumn_sizes_seq
      \@@_node_for_multicolumn:nn
  }
```

```
\cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
  {
    \cs_set_nopar:Npn \@@_i: { #1 }
    \cs_set_nopar:Npn \@@_j: { #2 }
  }
```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{`*n*`}{...}{...}` was issued in the format i-j and the second is the value of n (the length of the "multi-cell").

```
\cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
  {
    \@@_extract_coords_values: #1 \q_stop
    \@@_pgf_rect_node:nnnnn
      { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
      { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
      { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
      { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
      { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
    \str_if_empty:NF \l_@@_name_str
      {
        \pgfnodealias
          { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
          { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
      }
  }
```

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.

```
\keys_define:nn { NiceMatrix / Block / FirstPass }
  {
    l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
    l .value_forbidden:n = true ,
    r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
    r .value_forbidden:n = true ,
    c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
    c .value_forbidden:n = true ,
    L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
    L .value_forbidden:n = true ,
    R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
    R .value_forbidden:n = true ,
    C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
    C .value_forbidden:n = true ,
    t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
    t .value_forbidden:n = true ,
    b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
    b .value_forbidden:n = true ,
    color .tl_set:N = \l_@@_color_tl ,
    color .value_required:n = true ,
    respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
    respect-arraystretch .default:n = true ,
  }
```

The following command `\@@_Block:` will be linked to `\Block` in the environments of nicematrix. We define it with `\NewExpandableDocumentCommand` because it has an optional argument between `<` and `>`. It's mandatory to use an expandable command.

```
\NewExpandableDocumentCommand \@@_Block: { O { } m D < > { } +m }
  {
```

If the first mandatory argument of the command (which is the size of the block with the syntax i-j) has not be provided by the user, you use `1-1` (that is to say a block of only one cell).

```
    \peek_remove_spaces:n
```

```
      {
        \tl_if_blank:nTF { #2 }
          { \@@_Block_i 1-1 \q_stop }
          { \@@_Block_i #2 \q_stop }
        { #1 } { #3 } { #4 }
      }
  }
```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
\cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
```

Now, the arguments have been extracted: `#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of *key=values* pairs, `#4` are the tokens to put before the math mode and the beginning of the small array of the block and `#5` is the label of the block.

```
\cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
  {
```

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).

```
    \bool_lazy_or:nnTF
      { \tl_if_blank_p:n { #1 } }
      { \str_if_eq_p:nn { #1 } { * } }
      { \int_set:Nn \l_tmpa_int { 100 } }
      { \int_set:Nn \l_tmpa_int { #1 } }
    \bool_lazy_or:nnTF
      { \tl_if_blank_p:n { #2 } }
      { \str_if_eq_p:nn { #2 } { * } }
      { \int_set:Nn \l_tmpb_int { 100 } }
      { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
    \int_compare:nNnTF \l_tmpb_int = 1
      {
        \str_if_empty:NTF \l_@@_hpos_cell_str
          { \str_set:Nn \l_@@_hpos_block_str c }
          { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
      }
      { \str_set:Nn \l_@@_hpos_block_str c }
```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```
    \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
    \tl_set:Nx \l_tmpa_tl
      {
        { \int_use:N \c@iRow }
        { \int_use:N \c@jCol }
        { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
        { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
      }
```

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).

```
    \bool_if:nTF
```

```
      {
        (
          \int_compare_p:nNn { \l_tmpa_int } = 1
             ||
          \int_compare_p:nNn { \l_tmpb_int } = 1
        )
        && ! \tl_if_empty_p:n { #5 }
```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```
        && ! \l_@@_X_column_bool
      }
      { \exp_args:Nxx \@@_Block_iv:nnnnn }
      { \exp_args:Nxx \@@_Block_v:nnnnn }
    { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
  }
```

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).

```
\cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
  {
    \int_gincr:N \g_@@_block_box_int
    \cs_set_protected_nopar:Npn \diagbox ##1 ##2
      {
        \tl_gput_right:Nx \g_@@_internal_code_after_tl
          {
            \@@_actually_diagbox:nnnnnn
              { \int_use:N \c@iRow }
              { \int_use:N \c@jCol }
              { \int_eval:n { \c@iRow + #1 - 1 } }
              { \int_eval:n { \c@jCol + #2 - 1 } }
              { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
          }
      }
    \box_gclear_new:c
      { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
    \hbox_gset:cn
      { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
      {
```

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with \set@color and not \color_ensure_current: (in order to use \color_ensure_current: safely, you should load l3backend before the \documentclass with \RequirePackage{expl3}).

```
        \tl_if_empty:NTF \l_@@_color_tl
          { \int_compare:nNnT { #2 } = 1 \set@color }
          { \color { \l_@@_color_tl } }
```

If the block is mono-row, we use \g_@@_row_style_tl even if it has yet been used in the beginning of the cell where the command \Block has been issued because we want to be able to take into account a potential instruction of color of the font in \g_@@_row_style_tl.

```
        \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
        \group_begin:
        \bool_if:NF \l_@@_respect_arraystretch_bool
          { \cs_set:Npn \arraystretch { 1 } }
        \dim_zero:N \extrarowheight
        #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).

```
        \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
        \bool_if:NTF \l_@@_NiceTabular_bool
          {
            \bool_lazy_all:nTF
              {
                { \int_compare_p:nNn { #2 } = 1 }
                { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
                { ! \l_@@_respect_arraystretch_bool }
              }
```

When the block is mono-column in a column with a fixed width (eg p{3cm}).

```
              {
                \begin { minipage } [ \l_@@_vpos_of_block_tl ]
                  { \l_@@_col_width_dim }
                  \str_case:Vn \l_@@_hpos_block_str
                    {
                      c \centering
                      r \raggedleft
                      l \raggedright
                    }
                  #5
                \end { minipage }
              }
              {
                \use:x
                  {
                    \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
                      { @ { } \l_@@_hpos_block_str @ { } }
                  }
                  #5
                \end { tabular }
              }
          }
          {
            \c_math_toggle_token
            \use:x
              {
                \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
                { @ { } \l_@@_hpos_block_str @ { } }
              }
              #5
            \end { array }
            \c_math_toggle_token
          }
        \group_end:
      }
    \bool_if:NT \g_@@_rotate_bool
      {
        \box_grotate:cn
          { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
          { 90 }
        \bool_gset_false:N \g_@@_rotate_bool
      }
```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```
    \int_compare:nNnT { #2 } = 1
      {
        \dim_gset:Nn \g_@@_blocks_wd_dim
```

```
          {
            \dim_max:nn
              \g_@@_blocks_wd_dim
              {
                \box_wd:c
                  { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
              }
          }
      }
```

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.

```
    \int_compare:nNnT { #1 } = 1
      {
        \dim_gset:Nn \g_@@_blocks_ht_dim
          {
            \dim_max:nn
              \g_@@_blocks_ht_dim
              {
                \box_ht:c
                  { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
              }
          }
        \dim_gset:Nn \g_@@_blocks_dp_dim
          {
            \dim_max:nn
              \g_@@_blocks_dp_dim
              {
                \box_dp:c
                  { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
              }
          }
      }
    \seq_gput_right:Nx \g_@@_blocks_seq
      {
        \l_tmpa_tl
```

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in \l_@@_hpos_block_str. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of \l_@@_hpos_block_str, which is fixed by the type of current column.

```
        { \exp_not:n { #3 } , \l_@@_hpos_block_str }
        {
          \box_use_drop:c
            { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
        }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
  {
    \seq_gput_right:Nx \g_@@_blocks_seq
      {
        \l_tmpa_tl
        { \exp_not:n { #3 } }
        \exp_not:n
          {
            {
              \bool_if:NTF \l_@@_NiceTabular_bool
                {
                  \group_begin:
                  \bool_if:NF \l_@@_respect_arraystretch_bool
                    { \cs_set:Npn \arraystretch { 1 } }
                  \dim_zero:N \extrarowheight
                  #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).

```
                  \bool_if:NT \g_@@_rotate_bool
                    { \str_set:Nn \l_@@_hpos_block_str c }
                  \use:x
                    {
                      \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
                      { @ { } \l_@@_hpos_block_str @ { } }
                    }
                    #5
                  \end { tabular }
                  \group_end:
                }
                {
                  \group_begin:
                  \bool_if:NF \l_@@_respect_arraystretch_bool
                    { \cs_set:Npn \arraystretch { 1 } }
                  \dim_zero:N \extrarowheight
                  #4
                  \bool_if:NT \g_@@_rotate_bool
                    { \str_set:Nn \l_@@_hpos_block_str c }
                  \c_math_toggle_token
                  \use:x
                    {
                      \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
                      { @ { } \l_@@_hpos_block_str @ { } }
                    }
                    #5
                  \end { array }
                  \c_math_toggle_token
                  \group_end:
                }
            }
          }
      }
  }
```

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).

```
\keys_define:nn { NiceMatrix / Block / SecondPass }
  {
    tikz .code:n =
      \bool_if:NTF \c_@@_tikz_loaded_bool
        { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
        { \@@_error:n { tikz~key~without~tikz } } ,
    tikz .value_required:n = true ,
    fill .tl_set:N = \l_@@_fill_tl ,
    fill .value_required:n = true ,
    draw .tl_set:N = \l_@@_draw_tl ,
    draw .default:n = default ,
    rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
    rounded-corners .default:n = 4 pt ,
    color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
    color .value_required:n = true ,
    borders .clist_set:N = \l_@@_borders_clist ,
    borders .value_required:n = true ,
    hvlines .meta:n = { vlines , hlines } ,
    vlines .bool_set:N = \l_@@_vlines_block_bool,
    vlines .default:n = true ,
    hlines .bool_set:N = \l_@@_hlines_block_bool,
    hlines .default:n = true ,
    line-width .dim_set:N = \l_@@_line_width_dim ,
    line-width .value_required:n = true ,
    l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
    l .value_forbidden:n = true ,
    r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
    r .value_forbidden:n = true ,
    c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
    c .value_forbidden:n = true ,
    L .code:n = \str_set:Nn \l_@@_hpos_block_str l
                \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
    L .value_forbidden:n = true ,
    R .code:n = \str_set:Nn \l_@@_hpos_block_str r
                \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
    R .value_forbidden:n = true ,
    C .code:n = \str_set:Nn \l_@@_hpos_block_str c
                \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
    C .value_forbidden:n = true ,
    t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
    t .value_forbidden:n = true ,
    b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
    b .value_forbidden:n = true ,
    name .tl_set:N = \l_@@_block_name_str ,
    name .value_required:n = true ,
    name .initial:n = ,
    respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
    respect-arraystretch .default:n = true ,
    v-center .bool_set:N = \l_@@_v_center_bool ,
    v-center .default:n = true ,
    v-center .initial:n = false ,
    unknown .code:n = \@@_error:n { Unknown~key~for~Block }
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_blocks:
  {
    \cs_set_eq:NN \ialign \@@_old_ialign:
    \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
  }
```

```
\cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
  {
```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```
    \int_zero_new:N \l_@@_last_row_int
    \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.

```
    \int_compare:nNnTF { #3 } > { 99 }
      { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
      { \int_set:Nn \l_@@_last_row_int { #3 } }
    \int_compare:nNnTF { #4 } > { 99 }
      { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
      { \int_set:Nn \l_@@_last_col_int { #4 } }
    \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
      {
        \int_compare:nTF
          { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
          {
            \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
            \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
            \group_begin:
            \globaldefs = 1
            \@@_msg_redirect_name:nn { columns~not~used } { none }
            \group_end:
          }
          { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
      }
      {
        \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
          { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
          { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
      }
  }
```

```
\cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
  {
```

The group is for the keys.

```
    \group_begin:
    \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
```

We restrict the use of the key v-center to the case of a mono-row block.

```
    \bool_if:NT \l_@@_v_center_bool
      {
        \int_compare:nNnF { #1 } = { #3 }
          {
            \@@_error:n { Wrong~use~of~v-center }
            \bool_set_false:N \l_@@_v_center_bool
          }
      }
```

```
    \bool_if:NT \l_@@_vlines_block_bool
      {
        \tl_gput_right:Nx \g_nicematrix_code_after_tl
          {
            \@@_vlines_block:nnn
              { \exp_not:n { #5 } }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
          }
      }
    \bool_if:NT \l_@@_hlines_block_bool
      {
        \tl_gput_right:Nx \g_nicematrix_code_after_tl
          {
            \@@_hlines_block:nnn
              { \exp_not:n { #5 } }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
          }
      }
    \bool_if:nT
```

```
      { ! \l_@@_vlines_block_bool && ! \l_@@_hlines_block_bool }
      {
```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```
        \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
          { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
      }
    \tl_if_empty:NF \l_@@_draw_tl
      {
        \tl_gput_right:Nx \g_nicematrix_code_after_tl
          {
            \@@_stroke_block:nnn
              { \exp_not:n { #5 } }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
          }
        \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
           { { #1 } { #2 } { #3 } { #4 } }
      }
    \clist_if_empty:NF \l_@@_borders_clist
      {
        \tl_gput_right:Nx \g_nicematrix_code_after_tl
          {
            \@@_stroke_borders_block:nnn
              { \exp_not:n { #5 } }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
          }
      }
    \tl_if_empty:NF \l_@@_fill_tl
      {
```

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`.

```
        \exp_last_unbraced:NV \@@_extract_brackets \l_@@_fill_tl \q_stop
        \tl_gput_right:Nx \g_nicematrix_code_before_tl
          {
            \exp_not:N \roundedrectanglecolor
              [ \l_tmpa_tl ]
              { \exp_not:V \l_tmpb_tl }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
              { \dim_use:N \l_@@_rounded_corners_dim }
          }
      }
    \seq_if_empty:NF \l_@@_tikz_seq
      {
        \tl_gput_right:Nx \g_nicematrix_code_before_tl
          {
            \@@_block_tikz:nnnnn
              { #1 }
              { #2 }
              { \int_use:N \l_@@_last_row_int }
              { \int_use:N \l_@@_last_col_int }
              { \seq_use:Nn \l_@@_tikz_seq { , } }
          }
      }

    \cs_set_protected_nopar:Npn \diagbox ##1 ##2
      {
        \tl_gput_right:Nx \g_@@_internal_code_after_tl
```

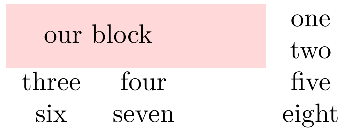
```
          {
            \@@_actually_diagbox:nnnnnn
              { #1 }
              { #2 }
              { \int_use:N \l_@@_last_row_int }
              { \int_use:N \l_@@_last_col_int }
              { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
          }
      }
```

```
    \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
    \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
```

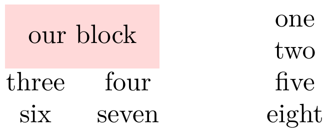
Let's consider the following `{NiceTabular}`. Because of the instruction `!{\hspace{1cm}}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the 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`



We highlight the node `1-1-block-short`



The construction of the node corresponding to the merged cells.

```
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_qpoint:n { row - #1 }
      \dim_set_eq:NN \l_tmpa_dim \pgf@y
      \@@_qpoint:n { col - #2 }
      \dim_set_eq:NN \l_tmpb_dim \pgf@x
      \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
      \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
      \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.

```
      \@@_pgf_rect_node:nnnnn
        { \@@_env: - #1 - #2 - block }
        \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
      \str_if_empty:NF \l_@@_block_name_str
        {
          \pgfnodealias
            { \@@_env: - \l_@@_block_name_str }
            { \@@_env: - #1 - #2 - block }
          \str_if_empty:NF \l_@@_name_str
            {
              \pgfnodealias
                { \l_@@_name_str - \l_@@_block_name_str }
                { \@@_env: - #1 - #2 - block }
            }
        }
```

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.

```
      \bool_if:NF \l_@@_hpos_of_block_cap_bool
        {
          \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.

```
          \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
            {
```

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.

```
              \cs_if_exist:cT
                { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
                {
                  \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
                    {
                      \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
                      \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
                    }
                }
            }
```

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.

```
          \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
            {
              \@@_qpoint:n { col - #2 }
              \dim_set_eq:NN \l_tmpb_dim \pgf@x
            }
          \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
          \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
            {
              \cs_if_exist:cT
                { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
                {
                  \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
                    {
                      \pgfpointanchor
                        { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
                        { east }
                      \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
                    }
                }
            }
          \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
            {
              \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
              \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
            }
          \@@_pgf_rect_node:nnnnn
            { \@@_env: - #1 - #2 - block - short }
            \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
        }
```

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.

```
    \bool_if:NT \l_@@_medium_nodes_bool
      {
        \@@_pgf_rect_node:nnn
```

```
          { \@@_env: - #1 - #2 - block - medium }
          { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
          {
            \pgfpointanchor
              { \@@_env:
                - \int_use:N \l_@@_last_row_int
                - \int_use:N \l_@@_last_col_int - medium
              }
              { south~east }
          }
      }
```

Now, we will put the label of the block beginning with the case of a \Block of one row.

```
    \bool_if:nTF
      { \int_compare_p:nNn { #1 } = { #3 } && ! \l_@@_v_center_bool }
      {
```

We take into account the case of a block of one row in the “first row” or the “last row”.

```
        \int_compare:nNnTF { #1 } = 0
          { \l_@@_code_for_first_row_tl }
          {
            \int_compare:nNnT { #1 } = \l_@@_last_row_int
              \l_@@_code_for_last_row_tl
          }
```

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.

```
            \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }
```

We retrieve (in \pgf@x) the x-value of the center of the block.

```
        \pgfpointanchor
          {
            \@@_env: - #1 - #2 - block
            \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
          }
          {
            \str_case:Vn \l_@@_hpos_block_str
              {
                c { center }
                l { west }
                r { east }
              }
          }
```

We put the label of the block which has been composed in \l_@@_cell_box.

```
        \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
        \pgfset { inner~sep = \c_zero_dim }
        \pgfnode
          { rectangle }
          {
             \str_case:Vn \l_@@_hpos_block_str
              {
                c { base }
                l { base~west }
                r { base~east }
              }
          }
          { \box_use_drop:N \l_@@_cell_box } { } { }
      }
```

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).

```
      {
```

If we are in the first column, we must put the block as if it was with the key `r`.

```
        \int_compare:nNnT { #2 } = 0
          { \str_set:Nn \l_@@_hpos_block_str r }
        \bool_if:nT \g_@@_last_col_found_bool
          {
            \int_compare:nNnT { #2 } = \g_@@_col_total_int
              { \str_set:Nn \l_@@_hpos_block_str l }
          }
        \pgftransformshift
          {
            \pgfpointanchor
              {
                \@@_env: - #1 - #2 - block
                \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
              }
              {
                \str_case:Vn \l_@@_hpos_block_str
                  {
                    c { center }
                    l { west }
                    r { east }
                  }
              }
          }
        \pgfset { inner~sep = \c_zero_dim }
        \pgfnode
          { rectangle }
          {
             \str_case:Vn \l_@@_hpos_block_str
              {
                c { center }
                l { west }
                r { east }
              }
          }
          { \box_use_drop:N \l_@@_cell_box } { } { }
      }
    \endpgfpicture
    \group_end:
  }
```

```
\NewDocumentCommand \@@_extract_brackets { O { } }
  {
    \tl_set:Nn \l_tmpa_tl { #1 }
    \@@_store_in_tmpb_tl
  }
\cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
  { \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).

```
\cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
  {
    \group_begin:
    \tl_clear:N \l_@@_draw_tl
    \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
    \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \tl_if_empty:NF \l_@@_draw_tl
```

```
   {
```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```
    \str_if_eq:VnTF \l_@@_draw_tl { default }
     { \CT@arc@ }
     { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
   }
  \pgfsetcornersarced
   {
    \pgfpoint
     { \dim_use:N \l_@@_rounded_corners_dim }
     { \dim_use:N \l_@@_rounded_corners_dim }
   }
  \@@_cut_on_hyphen:w #2 \q_stop
  \bool_lazy_and:nnT
   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
   {
    \@@_qpoint:n { row - \l_tmpa_tl }
    \dim_set:Nn \l_tmpb_dim { \pgf@y }
    \@@_qpoint:n { col - \l_tmpb_tl }
    \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
    \@@_cut_on_hyphen:w #3 \q_stop
    \int_compare:nNnT \l_tmpa_tl > \c@iRow
     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
    \int_compare:nNnT \l_tmpb_tl > \c@jCol
     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
    \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
    \dim_set:Nn \l_tmpa_dim { \pgf@y }
    \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
    \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
    \pgfpathrectanglecorners
     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
    \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```
    \pgfusepath { stroke }
   }
  \endpgfpicture
  \group_end:
 }
```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```
\keys_define:nn { NiceMatrix / BlockStroke }
 {
  color .tl_set:N = \l_@@_draw_tl ,
  draw .tl_set:N = \l_@@_draw_tl ,
  draw .default:n = default ,
  line-width .dim_set:N = \l_@@_line_width_dim ,
  rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
  rounded-corners .default:n = 4 pt
 }
```

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i*-*j*) and the third is the last cell of the block (with the same syntax).

```
\cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
 {
  \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
  \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
  \@@_cut_on_hyphen:w #2 \q_stop
  \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
```

```
    \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
    \@@_cut_on_hyphen:w #3 \q_stop
    \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
    \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
    \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
      {
        \use:x
          {
            \@@_vline:n
              {
                position = ##1 ,
                start = \l_@@_tmpc_tl ,
                end = \int_eval:n { \l_tmpa_tl - 1 }
              }
          }
      }
  }
\cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
  {
    \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
    \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
    \@@_cut_on_hyphen:w #2 \q_stop
    \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
    \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
    \@@_cut_on_hyphen:w #3 \q_stop
    \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
    \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
    \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
      {
        \use:x
          {
            \@@_hline:n
              {
                position = ##1 ,
                start = \l_@@_tmpd_tl ,
                end = \int_eval:n { \l_tmpb_tl - 1 }
              }
          }
      }
  }
```

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).

```
\cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
  {
    \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
    \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
    \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
      { \@@_error:n { borders~forbidden } }
      {
        \tl_clear_new:N \l_@@_borders_tikz_tl
        \keys_set:nV
          { NiceMatrix / OnlyForTikzInBorders }
          \l_@@_borders_clist
        \@@_cut_on_hyphen:w #2 \q_stop
        \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
        \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
        \@@_cut_on_hyphen:w #3 \q_stop
        \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
        \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
        \@@_stroke_borders_block_i:
      }
  }
```

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \cs_new_protected:Npx \@@_stroke_borders_block_i:
      {
        \c_@@_pgfortikzpicture_tl
        \@@_stroke_borders_block_ii:
        \c_@@_endpgfortikzpicture_tl
      }
  }
```

```
\cs_new_protected:Npn \@@_stroke_borders_block_ii:
  {
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \CT@arc@
    \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
    \clist_if_in:NnT \l_@@_borders_clist { right }
      { \@@_stroke_vertical:n \l_tmpb_tl }
    \clist_if_in:NnT \l_@@_borders_clist { left }
      { \@@_stroke_vertical:n \l_@@_tmpd_tl }
    \clist_if_in:NnT \l_@@_borders_clist { bottom }
      { \@@_stroke_horizontal:n \l_tmpa_tl }
    \clist_if_in:NnT \l_@@_borders_clist { top }
      { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
  }
```

```
\keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
  {
    tikz .code:n =
      \cs_if_exist:NTF \tikzpicture
        { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
        { \@@_error:n { tikz~in~borders~without~tikz } } ,
    tikz .value_required:n = true ,
    top .code:n = ,
    bottom .code:n = ,
    left .code:n = ,
    right .code:n = ,
    unknown .code:n = \@@_error:n { bad~border }
  }
```

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).

```
\cs_new_protected:Npn \@@_stroke_vertical:n #1
  {
    \@@_qpoint:n \l_@@_tmpc_tl
    \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
    \@@_qpoint:n \l_tmpa_tl
    \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
    \@@_qpoint:n { #1 }
    \tl_if_empty:NTF \l_@@_borders_tikz_tl
      {
        \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
        \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
        \pgfusepathqstroke
      }
      {
        \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
          ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
      }
  }
```

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).

```
\cs_new_protected:Npn \@@_stroke_horizontal:n #1
  {
    \@@_qpoint:n \l_@@_tmpd_tl
    \clist_if_in:NnTF \l_@@_borders_clist { left }
      { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
      { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
    \@@_qpoint:n \l_tmpb_tl
    \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
    \@@_qpoint:n { #1 }
    \tl_if_empty:NTF \l_@@_borders_tikz_tl
      {
        \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
        \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
        \pgfusepathqstroke
      }
      {
        \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
          ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
      }
  }
```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```
\keys_define:nn { NiceMatrix / BlockBorders }
  {
    borders .clist_set:N = \l_@@_borders_clist ,
    rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
    rounded-corners .default:n = 4 pt ,
    line-width .dim_set:N = \l_@@_line_width_dim ,
  }
```

The following command will be used if the key `tikz` has been used for the command `\Block`. The arguments `#1` and `#2` are the coordinates of the first cell and `#3` and `#4` the coordinates of the last cell of the block. `#5` is a comma-separated list of the Tikz keys used with the path.

```
\cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
  {
    \begin { tikzpicture }
    \clist_map_inline:nn { #5 }
      {
        \path [ ##1 ]
              ( #1 -| #2 )
              rectangle
              ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
      }
    \end { tikzpicture }
  }
```

How to draw the dotted lines transparently

```
\cs_set_protected:Npn \@@_renew_matrix:
  {
    \RenewDocumentEnvironment { pmatrix } { }
      { \pNiceMatrix }
      { \endpNiceMatrix }
    \RenewDocumentEnvironment { vmatrix } { }
      { \vNiceMatrix }
      { \endvNiceMatrix }
    \RenewDocumentEnvironment { Vmatrix } { }
      { \VNiceMatrix }
      { \endVNiceMatrix }
    \RenewDocumentEnvironment { bmatrix } { }
      { \bNiceMatrix }
      { \endbNiceMatrix }
    \RenewDocumentEnvironment { Bmatrix } { }
      { \BNiceMatrix }
      { \endBNiceMatrix }
  }
```

Automatic arrays

```
\cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
  {
    \int_set:Nn \l_@@_nb_rows_int { #1 }
    \int_set:Nn \l_@@_nb_cols_int { #2 }
  }
```

We will extract the potential keys l, r and c and pass the other keys to the environment {NiceArrayWithDelims}.

```
\keys_define:nn { NiceMatrix / Auto }
  {
    l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
    r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
    c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
  }
\NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
  {
    \int_zero_new:N \l_@@_nb_rows_int
    \int_zero_new:N \l_@@_nb_cols_int
    \@@_set_size:n #4 \q_stop
```

The group is for the protection of \l_@@_type_of_col_tl.

```
    \group_begin:
    \tl_set:Nn \l_@@_type_of_col_tl c
    \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
    \use:x
      {
        \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
          { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
          [ \exp_not:V \l_tmpa_tl ]
      }
    \int_compare:nNnT \l_@@_first_row_int = 0
      {
        \int_compare:nNnT \l_@@_first_col_int = 0 { & }
        \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
        \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
      }
    \prg_replicate:nn \l_@@_nb_rows_int
      {
        \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).

```
        \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
        \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
      }
    \int_compare:nNnT \l_@@_last_row_int > { -2 }
      {
        \int_compare:nNnT \l_@@_first_col_int = 0 { & }
        \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
        \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
      }
    \end { NiceArrayWithDelims }
    \group_end:
  }
```

```
\cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
  {
    \cs_set_protected:cpn { #1 AutoNiceMatrix }
      {
        \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
        \AutoNiceMatrixWithDelims { #2 } { #3 }
      }
  }
```

```
\@@_define_com:nnn p ( )
\@@_define_com:nnn b [ ]
\@@_define_com:nnn v | |
\@@_define_com:nnn V \| \|
\@@_define_com:nnn B \{ \}
```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```
\NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
  {
    \group_begin:
    \bool_set_true:N \l_@@_NiceArray_bool
    \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
    \group_end:
  }
```

The redefinition of the command \dotfill

```
\cs_set_eq:NN \@@_old_dotfill \dotfill
\cs_new_protected:Npn \@@_dotfill:
  {
```

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}`).

```
    \@@_old_dotfill
    \bool_if:NT \l_@@_NiceTabular_bool
      { \group_insert_after:N \@@_dotfill_ii: }
      { \group_insert_after:N \@@_dotfill_i: }
  }
\cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
\cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }
```

Now, if the box if not empty (unfornately, 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`.

```
\cs_new_protected:Npn \@@_dotfill_iii:
  { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim  \@@_old_dotfill }
```

The command \diagbox

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of nicematrix. However, there are also redefinitions of `\diagbox` in other circonstancies.

```
\cs_new_protected:Npn \@@_diagbox:nn #1 #2
  {
    \tl_gput_right:Nx \g_@@_internal_code_after_tl
      {
        \@@_actually_diagbox:nnnnnn
          { \int_use:N \c@iRow }
          { \int_use:N \c@jCol }
          { \int_use:N \c@iRow }
          { \int_use:N \c@jCol }
          { \exp_not:n { #1 } }
          { \exp_not:n { #2 } }
      }
```

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`.

```
    \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
      {
        { \int_use:N \c@iRow }
        { \int_use:N \c@jCol }
        { \int_use:N \c@iRow }
        { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
        { }
      }
  }
```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```
\cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
  {
    \pgfpicture
    \pgf@relevantforpicturesizefalse
    \pgfrememberpicturepositiononpagetrue
    \@@_qpoint:n { row - #1 }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - #2 }
    \dim_set_eq:NN \l_tmpb_dim \pgf@x
    \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
    \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
    \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
    \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
    \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
    {
```

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.

```
      \CT@arc@
      \pgfsetroundcap
      \pgfusepathqstroke
    }
    \pgfset { inner~sep = 1 pt }
    \pgfscope
    \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
    \pgfnode { rectangle } { south~west }
      {
        \begin { minipage } { 20 cm }
        \@@_math_toggle_token: #5 \@@_math_toggle_token:
        \end { minipage }
      }
      { }
      { }
    \endpgfscope
    \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
    \pgfnode { rectangle } { north~east }
      {
        \begin { minipage } { 20 cm }
        \raggedleft
        \@@_math_toggle_token: #6 \@@_math_toggle_token:
        \end { minipage }
      }
      { }
      { }
    \endpgfpicture
  }
```

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.

```
\keys_define:nn { NiceMatrix }
  {
    CodeAfter / rules .inherit:n = NiceMatrix / rules ,
    CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
  }
\keys_define:nn { NiceMatrix / CodeAfter }
  {
    sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
    sub-matrix .value_required:n = true ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
    rules .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
  }
```

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the "normal syntax". For the case of "light-syntax", see the definition of the environment `{@@-light-syntax}` on p. 123.

In the environments of nicematrix, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
\cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_ii:n }
```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_ii:n` which begins with `\\`.

```
\cs_new_protected:Npn \@@_CodeAfter_i: { \\ \omit \@@_CodeAfter_ii:n }
```

We have to catch everything until the end of the current environment (of nicematrix). First, we go until the next command `\end`.

```
\cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
  {
    \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
    \@@_CodeAfter_iv:n
  }
```

We catch the argument of the command `\end` (in `#1`).

```
\cs_new_protected:Npn \@@_CodeAfter_iv:n #1
  {
```

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.

```
    \str_if_eq:eeTF \@currenvir { #1 } { \end { #1 } }
```

If this is not the `\end` we are looking for, we put those tokens in `\g_nicematrix_code_after_tl` and we go on searching for the next command `\end` with a recursive call to the command `\@@_CodeAfter:n`.

```
      {
        \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
        \@@_CodeAfter_ii:n
      }
  }
```

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 colummn. 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.

```
\cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \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.

```
    \@@_qpoint:n { row - 1 }
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
    \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).

```
    \bool_if:nTF { #3 }
      { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
      { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
    \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
          {
            \pgfpointanchor
              { \@@_env: - ##1 - #2 }
              { \bool_if:nTF { #3 } { west } { east } }
            \dim_set:Nn \l_tmpa_dim
              { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
          }
      }
```

Now we can put the delimiter with a node of PGF.

```
    \pgfset { inner~sep = \c_zero_dim }
    \dim_zero:N \nulldelimiterspace
    \pgftransformshift
      {
        \pgfpoint
          { \l_tmpa_dim }
          { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
      }
    \pgfnode
      { rectangle }
      { \bool_if:nTF { #3 } { east } { west } }
      {
```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```
        \nullfont
        \c_math_toggle_token
        \tl_if_empty:NF \l_@@_delimiters_color_tl
          { \color { \l_@@_delimiters_color_tl } }
        \bool_if:nTF { #3 } { \left #1 } { \left . }
        \vcenter
          {
            \nullfont
            \hrule \@height
                   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
                   \@depth \c_zero_dim
                   \@width \c_zero_dim
          }
        \bool_if:nTF { #3 } { \right . } { \right #1 }
        \c_math_toggle_token
      }
      { }
      { }
    \endpgfpicture
  }
```

The command \SubMatrix

```
\keys_define:nn { NiceMatrix / sub-matrix }
  {
    extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
    extra-height .value_required:n = true ,
    left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
    left-xshift .value_required:n = true ,
    right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
    right-xshift .value_required:n = true ,
    xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
    xshift .value_required:n = true ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    slim .bool_set:N = \l_@@_submatrix_slim_bool ,
    slim .default:n = true ,
    hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
    hlines .default:n = all ,
    vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
    vlines .default:n = all ,
    hvlines .meta:n = { hlines, vlines } ,
    hvlines .value_forbidden:n = true ,
  }
\keys_define:nn { NiceMatrix }
  {
    SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
    CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
    NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
    NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
    pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
    NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
  }
```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```
\keys_define:nn { NiceMatrix / SubMatrix }
  {
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
    hlines .default:n = all ,
    vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
    vlines .default:n = all ,
    hvlines .meta:n = { hlines, vlines } ,
    hvlines .value_forbidden:n = true ,
    name .code:n =
      \tl_if_empty:nTF { #1 }
```

```
        { \@@_error:n { Invalid~name~format } }
        {
          \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
            {
              \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
                { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
                {
                  \str_set:Nn \l_@@_submatrix_name_str { #1 }
                  \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
                }
            }
            { \@@_error:n { Invalid~name~format } }
        } ,
    rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
    rules .value_required:n = true ,
    code .tl_set:N = \l_@@_code_tl ,
    code .value_required:n = true ,
    name .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
  }
```

```
\NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
  {
    \peek_remove_spaces:n
      {
        \@@_cut_on_hyphen:w #3 \q_stop
        \tl_clear_new:N \l_@@_tmpc_tl
        \tl_clear_new:N \l_@@_tmpd_tl
        \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
        \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
        \@@_cut_on_hyphen:w #2 \q_stop
        \seq_gput_right:Nx \g_@@_submatrix_seq
          { { \l_tmpa_tl } { \l_tmpb_tl } { \l_@@_tmpc_tl } { \l_@@_tmpd_tl } }
        \tl_gput_right:Nn \g_@@_internal_code_after_tl
          { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
      }
  }
```

In the internal `code-after` and in the `\CodeAfter` the following command `\@@_SubMatrix` will be linked to `\SubMatrix`.

- `#1` is the left delimiter;
- `#2` is the upper-left cell of the matrix with the format *i*-*j*;
- `#3` is the lower-right cell of the matrix with the format *i*-*j*;
- `#4` is the right delimiter;
- `#5` is the list of options of the command;
- `#6` is the potential subscript;
- `#7` is the potential superscript.

For explanations about the construction with rescanning of the preamble, see the documentation for the user command `\Cdots`.

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \tl_set:Nn \l_@@_argspec_tl { m m m m O { } E { _ ^ } { { } { } } }
    \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
    \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
      {
        \peek_remove_spaces:n
          {
            \@@_sub_matrix:nnnnnnn
              { #1 } { #2 } { #3 } { #4 }  { #5 } { #6 } { #7 }
          }
      }
  }
```

The following macro will compute `\l_@@_first_i_tl`, `\l_@@_first_j_tl`, `\l_@@_last_i_tl` and `\l_@@_last_j_tl` from the arguments of the command as provided by the user (for example `2-3` and `5-last`).

```
\cs_new_protected:Npn \@@_compute_i_j:nn #1 #2
  {
    \tl_clear_new:N \l_@@_first_i_tl
    \tl_clear_new:N \l_@@_first_j_tl
    \tl_clear_new:N \l_@@_last_i_tl
    \tl_clear_new:N \l_@@_last_j_tl
    \@@_cut_on_hyphen:w #1 \q_stop
    \tl_if_eq:NnTF \l_tmpa_tl { last }
      { \tl_set:NV \l_@@_first_i_tl \c@iRow }
      { \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl }
    \tl_if_eq:NnTF \l_tmpb_tl { last }
      { \tl_set:NV \l_@@_first_j_tl \c@jCol }
      { \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl }
    \@@_cut_on_hyphen:w #2 \q_stop
    \tl_if_eq:NnTF \l_tmpa_tl { last }
      { \tl_set:NV \l_@@_last_i_tl \c@iRow }
      { \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl }
    \tl_if_eq:NnTF \l_tmpb_tl { last }
      { \tl_set:NV \l_@@_last_j_tl \c@jCol }
      { \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl }
  }
```

```
\cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
  {
    \group_begin:
```

The four following token lists correspond to the position of the `\SubMatrix`.

```
    \@@_compute_i_j:nn { #2 } { #3 }
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
      { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
      { \@@_error:nn { Construct~too~large } { \SubMatrix } }
      {
        \str_clear_new:N \l_@@_submatrix_name_str
        \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgf@relevantforpicturesizefalse
        \pgfset { inner~sep = \c_zero_dim }
        \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
        \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
```

The last value of `\int_step_inline:nnn` is provided by currifycation.

```
        \bool_if:NTF \l_@@_submatrix_slim_bool
          { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
          { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
          {
            \cs_if_exist:cT
              { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
              {
                \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
                \dim_set:Nn \l_@@_x_initial_dim
                  { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
              }
            \cs_if_exist:cT
```

```
              { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
              {
                \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
                \dim_set:Nn \l_@@_x_final_dim
                  { \dim_max:nn \l_@@_x_final_dim \pgf@x }
              }
          }
        \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
          { \@@_error:nn { impossible~delimiter } { left } }
          {
            \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
              { \@@_error:nn { impossible~delimiter } { right } }
              { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
          }
        \endpgfpicture
      }
    \group_end:
  }
```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```
\cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
  {
    \@@_qpoint:n { row - \l_@@_first_i_tl - base }
    \dim_set:Nn \l_@@_y_initial_dim
      { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
    \@@_qpoint:n { row - \l_@@_last_i_tl - base }
    \dim_set:Nn \l_@@_y_final_dim
      { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
    \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
          {
            \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
            \dim_set:Nn \l_@@_y_initial_dim
              { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
          }
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
          {
            \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
            \dim_set:Nn \l_@@_y_final_dim
              { \dim_min:nn \l_@@_y_final_dim \pgf@y }
          }
      }
    \dim_set:Nn \l_tmpa_dim
      {
        \l_@@_y_initial_dim - \l_@@_y_final_dim +
        \l_@@_submatrix_extra_height_dim - \arrayrulewidth
      }
    \dim_zero:N \nulldelimiterspace
```

We will draw the rules in the \SubMatrix.

```
    \group_begin:
    \pgfsetlinewidth { 1.1 \arrayrulewidth }
    \tl_if_empty:NF \l_@@_rules_color_tl
      { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
    \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.

```
    \seq_map_inline:Nn \g_@@_cols_vlism_seq
      {
        \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
          {
            \int_compare:nNnT
              { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
              {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
                \@@_qpoint:n { col - ##1 }
                \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
                \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
                \pgfusepathqstroke
              }
          }
      }
```

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.

```
    \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
      { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
      { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
      {
        \bool_lazy_and:nnTF
          { \int_compare_p:nNn { ##1 } > 0 }
          {
             \int_compare_p:nNn
               { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
          {
            \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
            \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
            \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
            \pgfusepathqstroke
          }
          { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
      }
```

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.

```
    \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
      { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
      { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
      {
        \bool_lazy_and:nnTF
          { \int_compare_p:nNn { ##1 } > 0 }
          {
            \int_compare_p:nNn
              { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
          {
            \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```
            \group_begin:
```

We compute in `\l_tmpa_dim` the x-value of the left end of the rule.

```
            \dim_set:Nn \l_tmpa_dim
              { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
            \str_case:nn { #1 }
              {
                (  { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
                [  { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
                \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
              }
            \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```

We compute in `\l_tmpb_dim` the x-value of the right end of the rule.

```
            \dim_set:Nn \l_tmpb_dim
              { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
            \str_case:nn { #2 }
              {
                )  { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
                ]  { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
                \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
              }
            \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
            \pgfusepathqstroke
            \group_end:
          }
          { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
      }
```

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).

```
    \str_if_empty:NF \l_@@_submatrix_name_str
      {
        \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
          \l_@@_x_initial_dim \l_@@_y_initial_dim
          \l_@@_x_final_dim \l_@@_y_final_dim
      }
    \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`.

```
    \begin { pgfscope }
    \pgftransformshift
      {
        \pgfpoint
          { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
          { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
      }
    \str_if_empty:NTF \l_@@_submatrix_name_str
      { \@@_node_left:nn #1 { } }
      { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
    \end { pgfscope }
```

Now, we deal with the right delimiter.

```
    \pgftransformshift
      {
        \pgfpoint
          { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
          { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
      }
    \str_if_empty:NTF \l_@@_submatrix_name_str
      { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
      {
        \@@_node_right:nnnn #2
          { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
      }
    \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
    \flag_clear_new:n { nicematrix }
    \l_@@_code_tl
  }
```

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, `row-`i, `col-`j and i`-|`j refer to the number of row and columm *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```
\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.

```
\cs_new_protected:Npn \@@_pgfpointanchor:n #1
  {
    \use:e
      { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
  }
```

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 `-`.

```
\cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
  { #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.

```
\tl_const:Nn \c_@@_integers_alist_tl
  {
    { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
    { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
    { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
    { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
  }
```

```
\cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
  {
```

If there is no hyphen, that means that the node is of the form of a single number (ex.: `5` or `11`). In that case, we are in an analysis which result from a specification of node of the form *i*`-|`*j*. In that case, the *i* of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the *j* arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```
    \tl_if_empty:nTF { #2 }
      {
        \str_case:nVTF { #1 } \c_@@_integers_alist_tl
          {
            \flag_raise:n { nicematrix }
            \int_if_even:nTF { \flag_height:n { nicematrix } }
              { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
              { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
          }
          { #1 }
      }
```

If there is an hyphen, we have to see whether we have a node of the form *i*-*j*, `row-`*i* or `col-`*j*.

```
      { \@@_pgfpointanchor_iii:w { #1 } #2 }
  }
```

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`).

```
\cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
```

```
  {
    \str_case:nnF { #1 }
      {
        { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
        { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
      }
```

Now the case of a node of the form i-j.

```
      {
        \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
        - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
      }
  }
```

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`).

```
\cs_new_protected:Npn \@@_node_left:nn #1 #2
  {
    \pgfnode
      { rectangle }
      { east }
      {
        \nullfont
        \c_math_toggle_token
        \tl_if_empty:NF \l_@@_delimiters_color_tl
          { \color { \l_@@_delimiters_color_tl } }
        \left #1
        \vcenter
          {
            \nullfont
            \hrule \@height \l_tmpa_dim
                   \@depth \c_zero_dim
                   \@width \c_zero_dim
          }
        \right .
        \c_math_toggle_token
      }
      { #2 }
      { }
  }
```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument `#3` is the subscript and `#4` is the superscript.

```
\cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
  {
    \pgfnode
      { rectangle }
      { west }
      {
        \nullfont
        \c_math_toggle_token
        \tl_if_empty:NF \l_@@_delimiters_color_tl
          { \color { \l_@@_delimiters_color_tl } }
        \left .
        \vcenter
          {
            \nullfont
            \hrule \@height \l_tmpa_dim
                   \@depth \c_zero_dim
                   \@width \c_zero_dim
          }
        \right #1
        \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
        ^ { \smash { #4 } }
        \c_math_toggle_token
      }
      { #2 }
      { }
  }
```

Les commandes \UnderBrace et \OverBrace

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```
\NewDocumentCommand \@@_UnderBrace { O { } m m m O { } }
  {
    \peek_remove_spaces:n
      { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
  }
\NewDocumentCommand \@@_OverBrace { O { } m m m O { } }
  {
    \peek_remove_spaces:n
      { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
  }
```

```
\keys_define:nn { NiceMatrix / Brace }
  {
    left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
    left-shorten .default:n = true ,
    right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
    shorten .meta:n = { left-shorten , right-shorten } ,
    right-shorten .default:n = true ,
    yshift .dim_set:N = \l_@@_brace_yshift_dim ,
    yshift .value_required:n = true ,
    yshift .initial:n = \c_zero_dim ,
    color .tl_set:N = \l_tmpa_tl ,
    color .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
  }
```

`#1` is the first cell of the rectangle (with the syntax i-|j; `#2` is the last cell of the rectangle; `#3` is the label of the text; `#4` is the optional argument (a list of *key-value* pairs); `#5` is equal to `under` or `over`.

```
\cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
  {
    \group_begin:
```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```
    \@@_compute_i_j:nn { #1 } { #2 }
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
      { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
      {
        \str_if_eq:nnTF { #5 } { under }
          { \@@_error:nn { Construct~too~large } { \UnderBrace } }
          { \@@_error:nn { Construct~too~large } { \OverBrace } }
      }
      {
        \tl_clear:N \l_tmpa_tl % added the 2022-02-25
        \keys_set:nn { NiceMatrix / Brace } { #4 }
        \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } } % added the 2022-02-25
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgf@relevantforpicturesizefalse
        \bool_if:NT \l_@@_brace_left_shorten_bool
          {
            \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
            \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
              {
                \cs_if_exist:cT
                  { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
                  {
                    \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
                    \dim_set:Nn \l_@@_x_initial_dim
                      { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
                  }
              }
          }
        \bool_lazy_or:nnT
          { \bool_not_p:n \l_@@_brace_left_shorten_bool }
          { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
          {
            \@@_qpoint:n { col - \l_@@_first_j_tl }
            \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
          }
        \bool_if:NT \l_@@_brace_right_shorten_bool
          {
            \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
            \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
              {
                \cs_if_exist:cT
                  { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
                  {
                    \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
                    \dim_set:Nn \l_@@_x_final_dim
                      { \dim_max:nn \l_@@_x_final_dim \pgf@x }
                  }
              }
          }
        \bool_lazy_or:nnT
          { \bool_not_p:n \l_@@_brace_right_shorten_bool }
          { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
          {
            \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
            \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
          }
        \pgfset { inner~sep = \c_zero_dim }
        \str_if_eq:nnTF { #5 } { under }
          { \@@_underbrace_i:n { #3 } }
          { \@@_overbrace_i:n { #3 } }
        \endpgfpicture
      }
    \group_end:
  }
```

The argument is the text to put above the brace.

```
\cs_new_protected:Npn \@@_overbrace_i:n #1
  {
    \@@_qpoint:n { row - \l_@@_first_i_tl }
    \pgftransformshift
      {
        \pgfpoint
          { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
          { \pgf@y + \l_@@_brace_yshift_dim }
      }
    \pgfnode
```

```
      { rectangle }
      { south }
      {
        \vbox_top:n
          {
            \group_begin:
            \everycr { }
            \halign
              {
                \hfil ## \hfil \crcr
                \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
                \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
                \hbox_to_wd:nn
                  { \l_@@_x_final_dim - \l_@@_x_initial_dim }
                  { \downbracefill } \cr
              }
            \group_end:
          }
      }
      { }
      { }
  }
```

The argument is the text to put under the brace.

```
\cs_new_protected:Npn \@@_underbrace_i:n #1
  {
    \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
    \pgftransformshift
      {
        \pgfpoint
          { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
          { \pgf@y  - \l_@@_brace_yshift_dim }
      }
    \pgfnode
      { rectangle }
      { north }
      {
        \group_begin:
        \everycr { }
        \vbox:n
          {
            \halign
              {
                \hfil ## \hfil \crcr
                \hbox_to_wd:nn
                  { \l_@@_x_final_dim - \l_@@_x_initial_dim }
                  { \upbracefill } \cr
                \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
                \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
              }
          }
        \group_end:
      }
      { }
      { }
  }
```

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.

```
\bool_new:N \c_@@_footnotehyper_bool
```

The boolean \c_@@_footnote_bool will indicate if the option footnote is used, but quicky, it will also be set to true if the option footnotehyper is used.

```
\bool_new:N \c_@@_footnote_bool
\@@_msg_new:nnn { Unknown~key~for~package }
  {
    The~key~'\l_keys_key_str'~is~unknown. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    allow-letter-for-dotted-lines,~
    footnote,~
    footnotehyper,~
    renew-dots,~and
    renew-matrix.
  }
\keys_define:nn { NiceMatrix / Package }
  {
    renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
    renew-dots .value_forbidden:n = true ,
    renew-matrix .code:n = \@@_renew_matrix: ,
    renew-matrix .value_forbidden:n = true ,
    transparent .code:n = \@@_fatal:n { Key~transparent } ,
    transparent .value_forbidden:n = true,
    footnote .bool_set:N = \c_@@_footnote_bool ,
    footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
    allow-letter-for-dotted-lines .code:n =
     {
       \group_begin:
       \globaldefs = 1
       \@@_msg_redirect_name:nn { letter-for-dotted-lines } { none }
       \group_end:
     } ,
    allow-letter-for-dotted-lines .value_forbidden:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~package }
  }
\ProcessKeysOptions { NiceMatrix / Package }
```

```
\@@_msg_new:nn { footnote~with~footnotehyper~package }
  {
    You~can't~use~the~option~'footnote'~because~the~package~
    footnotehyper~has~already~been~loaded.~
    If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
    within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
    of~the~package~footnotehyper.\\
    If~you~go~on,~the~package~footnote~won't~be~loaded.
  }
\@@_msg_new:nn { footnotehyper~with~footnote~package }
  {
    You~can't~use~the~option~'footnotehyper'~because~the~package~
    footnote~has~already~been~loaded.~
    If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
    within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
    of~the~package~footnote.\\
    If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
  }
```

```
\bool_if:NT \c_@@_footnote_bool
  {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
    \@ifclassloaded { beamer }
      { \bool_set_false:N \c_@@_footnote_bool }
      {
        \@ifpackageloaded { footnotehyper }
          { \@@_error:n { footnote~with~footnotehyper~package } }
          { \usepackage { footnote } }
      }
  }
\bool_if:NT \c_@@_footnotehyper_bool
  {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
    \@ifclassloaded { beamer }
      { \bool_set_false:N \c_@@_footnote_bool }
      {
        \@ifpackageloaded { footnote }
          { \@@_error:n { footnotehyper~with~footnote~package } }
          { \usepackage { footnotehyper } }
      }
    \bool_set_true:N \c_@@_footnote_bool
  }
```

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

```
\seq_new:N \g_@@_types_of_matrix_seq
\seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
  {
    NiceMatrix ,
    pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
  }
\seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
  { \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`.

```
\cs_new_protected:Npn \@@_error_too_much_cols:
  {
    \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
      {
        \int_compare:nNnTF \l_@@_last_col_int = { -2 }
        { \@@_fatal:n { too~much~cols~for~matrix } }
        {
          \bool_if:NF \l_@@_last_col_without_value_bool
            { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
        }
      }
      { \@@_fatal:n { too~much~cols~for~array } }
  }
```

The following command must *not* be protected since it's used in an error message.

```
\cs_new:Npn \@@_message_hdotsfor:
  {
    \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
      { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
  }
\@@_msg_new:nn { negative~weight }
  {
    The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
    the~value~'#1'.~If~you~go~on,~the~absolute~value~will~be~used.
  }
\@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
  {
    You~try~to~use~more~columns~than~allowed~by~your~
    \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
    columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
    exterior~columns).~This~error~is~fatal.
  }
\@@_msg_new:nn { too~much~cols~for~matrix }
  {
    You~try~to~use~more~columns~than~allowed~by~your~
    \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
    number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
    'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
    This~error~is~fatal.
  }
```

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`.

```
\@@_msg_new:nn { too~much~cols~for~array }
  {
    You~try~to~use~more~columns~than~allowed~by~your~
    \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
    \int_use:N \g_@@_static_num_of_col_int\
    ~(plus~the~potential~exterior~ones).~
    This~error~is~fatal.
  }
\@@_msg_new:nn { hvlines-except-corners }
  {
    The~key~'hvlines-except-corners'~is~now~obsolete.~You~should~instead~use~the~
    keys~'hvlines'~and~'corners'.\\
    However,~you~can~go~on~for~this~time.~This~message~won't~be~shown~anymore~
    in~this~document.
  }
\@@_msg_new:nn { last~col~not~used }
  {
    The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
    in~your~\@@_full_name_env:.~However,~you~can~go~on.
  }
\@@_msg_new:nn { columns~not~used }
  {
    The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
    \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
    You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
  }
\@@_msg_new:nn { in~first~col }
  {
    You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { in~last~col }
  {
    You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { in~first~row }
  {
    You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { in~last~row }
  {
    You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { double~closing~delimiter }
  {
    You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
    delimiter.~This~delimiter~will~be~ignored.
  }
\@@_msg_new:nn { delimiter~after~opening }
  {
    You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
    delimiter.~This~delimiter~will~be~ignored.
  }
\@@_msg_new:nn { bad~option~for~line-style }
  {
    Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
    is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
  }
\@@_msg_new:nnn { Unknown~key~for~custom-line }
  {
    The~key~'\l_keys_key_str'~is~unknown~in~a~'custom-line'.~
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    color,~
    command,~
    dotted,~
    letter,~
    multiplicity,~
    sep-color,~
    tikz,~and~width.
  }
\@@_msg_new:nn { Unknown~key~for~xdots }
  {
    As~for~now,~there~is~only~three~keys~available~here:~'color',~'line-style'~
    and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
    this~key~will~be~ignored.
  }
\@@_msg_new:nn { Unknown~key~for~rowcolors }
  {
    As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
    (and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
    this~key~will~be~ignored.
  }
```

```
\@@_msg_new:nn { ampersand~in~light-syntax }
  {
    You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
    ~you~have~used~the~key~'light-syntax'.~This~error~is~fatal.
  }
\@@_msg_new:nn { Construct~too~large }
  {
    Your~command~\token_to_str:N #1
    can't~be~drawn~because~your~matrix~is~too~small.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { double-backslash~in~light-syntax }
  {
    You~can't~use~\token_to_str:N \\~to~separate~rows~because~you~have~used~
    the~key~'light-syntax'.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
    (set~by~the~key~'end-of-row').~This~error~is~fatal.
  }
\@@_msg_new:nn { standard-cline~in~document }
  {
    The~key~'standard-cline'~is~available~only~in~the~preamble.\\
    If~you~go~on~this~command~will~be~ignored.
  }
\@@_msg_new:nn { bad~value~for~baseline }
  {
    The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
    valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
    \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
    If~you~go~on,~a~value~of~1~will~be~used.
  }
\@@_msg_new:nn { Invalid~name~format }
  {
    You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
    \SubMatrix.\\
    A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
    If~you~go~on,~this~key~will~be~ignored.
  }
\@@_msg_new:nn { Wrong~line~in~SubMatrix }
  {
    You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
    \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
    number~is~not~valid.~If~you~go~on,~it~will~be~ignored.
  }
\@@_msg_new:nn { impossible~delimiter }
  {
    It's~impossible~to~draw~the~#1~delimiter~of~your~
    \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
    in~that~column.
    \bool_if:NT \l_@@_submatrix_slim_bool
      { ~Maybe~you~should~try~without~the~key~'slim'. } \\
    If~you~go~on,~this~\token_to_str:N \SubMatrix\ will~be~ignored.
  }
\@@_msg_new:nn { width~without~X~columns }
  {
    You~have~used~the~key~'width'~but~you~have~put~no~'X'~column. \\
    If~you~go~on,~that~key~will~be~ignored.
  }
\@@_msg_new:nn { empty~environment }
  { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }
\@@_msg_new:nn { Wrong~use~of~v-center }
```

```
  {
    You~should~not~use~the~key~'v-center'~here~because~your~block~is~not~
    mono-row.~However,~you~can~go~on.
  }
\@@_msg_new:nn { No~letter~and~no~command }
  {
    Your~use~of~'custom-line'~is~no-op~since~you~don't~have~used~the~
    key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~key~'command'~
    (to~draw~horizontal~rules).\\
    However,~you~can~go~on.
  }
\@@_msg_new:nn { letter-for-dotted-lines }
  {
    The~key~'letter-for-dotted-lines'~is~now~obsolete~(you~should~
    use~'custom-line'~instead).~However,~you~can~go~on~for~this~time.~
    If~you~don't~want~to~see~that~message~again,~you~should~
    load~'nicematrix'~with~the~key~'allow-letter-for-dotted-lines'.~
    However,~'letter-for-dotted-lines'~will~be~deleted~in~a~future~
    version~of~'nicematrix'.
  }
\@@_msg_new:nn { Forbidden~letter }
  {
    You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
    If~you~go~on,~it~will~be~ignored.
  }
\@@_msg_new:nn { key~width~without~key~tikz }
  {
    In~'custom-line',~you~have~used~'width'~without~'tikz'.~That's~not~correct.~
    If~you~go~on,~that~key~'width'~will~be~discarded.
  }
\@@_msg_new:nn { Several~letters }
  {
    You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~
    have~used~'\l_@@_letter_str').\\
    If~you~go~on,~it~will~be~ignored.
  }
\@@_msg_new:nn { Delimiter~with~small }
  {
    You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
    because~the~key~'small'~is~in~force.\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
  {
    Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code-after'~
    can't~be~executed~because~a~cell~doesn't~exist.\\
    If~you~go~on~this~command~will~be~ignored.
  }
\@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
  {
    The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
    in~this~\@@_full_name_env:.\\
    If~you~go~on,~this~key~will~be~ignored.\\
    For~a~list~of~the~names~already~used,~type~H~<return>.
  }
  {
    The~names~already~defined~in~this~\@@_full_name_env:\ are:~
    \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
  }
\@@_msg_new:nn { r~or~l~with~preamble }
```

```
  {
    You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:.~
    You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
    your~\@@_full_name_env:.\\
    If~you~go~on,~this~key~will~be~ignored.
  }
\@@_msg_new:nn { Hdotsfor~in~col~0 }
  {
    You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
    the~array.~This~error~is~fatal.
  }
\@@_msg_new:nn { bad~corner }
  {
    #1~is~an~incorrect~specification~for~a~corner~(in~the~keys~
    'corners'~and~'except-corners').~The~available~
    values~are:~NW,~SW,~NE~and~SE.\\
    If~you~go~on,~this~specification~of~corner~will~be~ignored.
  }
\@@_msg_new:nn { bad~border }
  {
    \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
    (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
    The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
    also~use~the~key~'tikz'
    \bool_if:nF \c_@@_tikz_loaded_bool
      {~if~you~load~the~LaTeX~package~'tikz'}).\\
    If~you~go~on,~this~specification~of~border~will~be~ignored.
  }
\@@_msg_new:nn { tikz~key~without~tikz }
  {
    You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
    \Block'~because~you~have~not~loaded~Tikz.~
    If~you~go~on,~this~key~will~be~ignored.
  }
\@@_msg_new:nn { last-col~non~empty~for~NiceArray }
  {
    In~the~\@@_full_name_env:,~you~must~use~the~key~
    'last-col'~without~value.\\
    However,~you~can~go~on~for~this~time~
    (the~value~'\l_keys_value_tl'~will~be~ignored).
  }
\@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
  {
    In~\NiceMatrixoptions,~you~must~use~the~key~
    'last-col'~without~value.\\
    However,~you~can~go~on~for~this~time~
    (the~value~'\l_keys_value_tl'~will~be~ignored).
  }
\@@_msg_new:nn { Block~too~large~1 }
  {
    You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
    too~small~for~that~block. \\
  }
\@@_msg_new:nn { Block~too~large~2 }
  {
    The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
    \g_@@_static_num_of_col_int\
    columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
    specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
    (&)~at~the~end~of~the~first~row~of~your~
    \@@_full_name_env:.\\
    If~you~go~on,this~block~and~maybe~others~will~be~ignored.
  }
\@@_msg_new:nn { unknown~column~type }
  {
    The~column~type~'#1'~in~your~\@@_full_name_env:\
    is~unknown. \\
    This~error~is~fatal.
  }
\@@_msg_new:nn { colon~without~arydshln }
  {
    The~column~type~':'~in~your~\@@_full_name_env:\
    is~unknown.~If~you~want~to~use~':'~of~'arydshln',~you~should~
    load~that~package.~If~you~want~a~dotted~line~of~'nicematrix',~you~
    should~use~'\l_@@_letter_for_dotted_lines_str'.\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { tabularnote~forbidden }
  {
    You~can't~use~the~command~\token_to_str:N\tabularnote\
    ~in~a~\@@_full_name_env:.~This~command~is~available~only~in~
    \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { borders~forbidden }
  {
    You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
    because~the~option~'rounded-corners'~
    is~in~force~with~a~non-zero~value.\\
    If~you~go~on,~this~key~will~be~ignored.
  }
\@@_msg_new:nn { bottomrule~without~booktabs }
  {
    You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
    loaded~'booktabs'.\\
    If~you~go~on,~this~key~will~be~ignored.
  }
\@@_msg_new:nn { enumitem~not~loaded }
  {
    You~can't~use~the~command~\token_to_str:N\tabularnote\
    ~because~you~haven't~loaded~'enumitem'.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { tikz~in~custom-line~without~tikz }
  {
    You~have~used~the~key~'tikz'~in~the~definition~of~a~
    customized~line~(with~'custom-line')~but~Tikz~is~not~loaded.~
    You~can~go~on~but~you~will~have~another~error~if~you~actually~
    use~that~custom~line.
  }
\@@_msg_new:nn { tikz~in~borders~without~tikz }
  {
    You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
    command~'\token_to_str:N\Block')~but~Tikz~is~not~loaded.~
    If~you~go~on,~that~key~will~be~ignored.
  }
\@@_msg_new:nn { color~in~custom-line~with~tikz }
  {
    In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
    which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
    If~you~go~on,~the~key~'color'~will~be~discarded.
  }
\@@_msg_new:nn { Wrong~last~row }
  {
    You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
    \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
    If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
    last~row.~You~can~avoid~this~problem~by~using~'last-row'~
    without~value~(more~compilations~might~be~necessary).
  }
\@@_msg_new:nn { Yet~in~env }
  { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }
\@@_msg_new:nn { Outside~math~mode }
  {
    The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
    (and~not~in~\token_to_str:N \vcenter).\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { One~letter~allowed }
  {
    The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
    If~you~go~on,~it~will~be~ignored.
  }
\@@_msg_new:nn { varwidth~not~loaded }
  {
    You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
    loaded.\\
    If~you~go~on,~your~column~will~behave~like~'p'.
  }
\@@_msg_new:nnn { Unknown~key~for~Block }
  {
    The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
    \Block.\\ If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
    hlines,~hvlines,~l,~line-width,~name,~rounded-corners,~r,~respect-arraystretch,
    ~t,~tikz~and~vlines.
  }
\@@_msg_new:nn { Version~of~siunitx~too~old }
  {
    You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
    is~too~old.~You~need~at~least~v~3.0.\\
    This~error~is~fatal.
  }
\@@_msg_new:nnn { Unknown~key~for~Brace }
  {
    The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
    \UnderBrace\ and~\token_to_str:N \OverBrace.\\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
    right-shorten,~shorten~(which~fixes~both~left-shorten~and~
    right-shorten)~and~yshift.
  }
\@@_msg_new:nnn { Unknown~key~for~CodeAfter }
  {
    The~key~'\l_keys_key_str'~is~unknown.\\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~keys~in~\token_to_str:N
    \CodeAfter,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    delimiters/color,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    sub-matrix~(several~subkeys)~
    and~xdots~(several~subkeys).~
    The~latter~is~for~the~command~\token_to_str:N \line.
  }
\@@_msg_new:nnn { Unknown~key~for~SubMatrix }
  {
    The~key~'\l_keys_key_str'~is~unknown.\\
    If~you~go~on,~this~key~will~be~ignored. \\
    For~a~list~of~the~available~keys~in~\token_to_str:N
    \SubMatrix,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    'delimiters/color',~
    'extra-height',~
    'hlines',~
    'hvlines',~
    'left-xshift',~
    'name',~
    'right-xshift',~
    'rules'~(with~the~subkeys~'color'~and~'width'),~
    'slim',~
    'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
    and~'right-xshift').\\
  }
\@@_msg_new:nnn { Unknown~key~for~notes }
  {
    The~key~'\l_keys_key_str'~is~unknown.\\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    bottomrule,~
    code-after,~
    code-before,~
    detect-duplicates,~
    enumitem-keys,~
    enumitem-keys-para,~
    para,~
    label-in-list,~
    label-in-tabular~and~
    style.
  }
\@@_msg_new:nnn { Unknown~key~for~RowStyle }
  {
    The~key~'\l_keys_key_str'~is~unknown~for~the~command~
    \token_to_str:N \RowStyle. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    'bold',~
    'cell-space-top-limit',~
    'cell-space-bottom-limit',~
    'cell-space-limits',~
    'color',~
    'nb-rows'~and~
    'rowcolor'.
  }
\@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
  {
    The~key~'\l_keys_key_str'~is~unknown~for~the~command~
    \token_to_str:N \NiceMatrixOptions. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    allow-duplicate-names,~
    cell-space-bottom-limit,~
    cell-space-limits,~
    cell-space-top-limit,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    corners,~
    custom-key,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    delimiters~(several~subkeys),~
    end-of-row,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    notes~(several~subkeys),~
    nullify-dots,~
    renew-dots,~
    renew-matrix,~
    respect-arraystretch,~
    right-margin,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    small,~
    sub-matrix~(several~subkeys),
    vlines,~
    xdots~(several~subkeys).
  }
\@@_msg_new:nnn { Unknown~key~for~NiceArray }
  {
    The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
    \{NiceArray\}. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    b,~
    baseline,~
    c,~
    cell-space-bottom-limit,~
    cell-space-limits,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    colortbl-like,~
    columns-width,~
    corners,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    delimiters/color,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    name,~
    notes/bottomrule,~
    notes/para,~
    nullify-dots,~
    renew-dots,~
    respect-arraystretch,~
    right-margin,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    small,~
    t,~
    tabularnote,~
    vlines,~
    xdots/color,~
    xdots/shorten~and~
    xdots/line-style.
  }
```

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`).

```
\@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
  {
    The~key~'\l_keys_key_str'~is~unknown~for~the~
    \@@_full_name_env:. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    b,~
    baseline,~
    c,~
    cell-space-bottom-limit,~
    cell-space-limits,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    colortbl-like,~
    columns-width,~
    corners,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    delimiters~(several~subkeys),~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    l,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    name,~
    nullify-dots,~
    r,~
    renew-dots,~
    respect-arraystretch,~
    right-margin,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    small,~
    t,~
    vlines,~
    xdots/color,~
    xdots/shorten~and~
    xdots/line-style.
  }
\@@_msg_new:nnn { Unknown~key~for~NiceTabular }
  {
    The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
    \{NiceTabular\}. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    b,~
    baseline,~
    c,~
    cell-space-bottom-limit,~
    cell-space-limits,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    colortbl-like,~
    columns-width,~
    corners,~
    custom-line,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    name,~
    notes/bottomrule,~
    notes/para,~
    nullify-dots,~
    renew-dots,~
    respect-arraystretch,~
    right-margin,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    t,~
    tabularnote,~
    vlines,~
    xdots/color,~
    xdots/shorten~and~
    xdots/line-style.
  }
\@@_msg_new:nnn { Duplicate~name }
  {
    The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
    the~same~environment~name~twice.~You~can~go~on,~but,~
    maybe,~you~will~have~incorrect~results~especially~
    if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
    message~again,~use~the~key~'allow-duplicate-names'~in~
    '\token_to_str:N \NiceMatrixOptions'.\\
    For~a~list~of~the~names~already~used,~type~H~<return>. \\
  }
  {
    The~names~already~defined~in~this~document~are:~
    \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
  }
\@@_msg_new:nn { Option~auto~for~columns-width }
  {
    You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
    If~you~go~on,~the~key~will~be~ignored.
  }
```

19 History

The successive versions of the file `nicematrix.sty` provided by TeXLive are available on the SVN server of TeXLive:

`https:www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty`

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types `L`, `C` and `R`.

Changes between version 1.2 and 1.3

New environment `{pNiceArrayC}` and its variants.
Correction of a bug in the definition of `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` (in fact, it was a typo).
Options are now available locally in `{pNiceMatrix}` and its variants.
The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types `w` and `W` can now be used in the environments `{NiceArray}`, `{pNiceArrayC}` and its variants with the same meaning as in the package array.
New option `columns-width` to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of nicematrix were focused on the continuous dotted lines whereas the version 2.0 of nicematrix provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment `{pNiceArrayRC}`. With this new implementation, there is no restriction on the width of the columns.
The package nicematrix no longer loads mathtools but only amsmath.
Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option `code-for-first-row` is now available in the command `\NiceMatrixOptions`.
Following a discussion on TeX StackExchange[72], Tikz externalization is now deactivated in the environments of the package nicematrix.[73]

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{array}{cc}
\begin{array}{ccc} & C_j & \end{array} & \\
\begin{pmatrix} 0 & \vdots & 0 \\ & a\cdots & \cdots \\ 0 & & 0 \end{pmatrix} & L_i
\end{array}$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `O { }` in commands and environments defined with xparse by `! O { }` (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`

[72] cf. `tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package`

[73] Before this version, there was an error when using nicematrix with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by nicematrix because they use the options `overlay` and `remember picture`.

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes revtex4-1 and revtex4-2.
Option `allow-duplicate-names`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of arydshln).
Possibility to draw vertical dotted lines to separate columns with the specifier "`:`" in the preamble (similar to the classical specifier "`|`" and the specifier "`:`" of arydshln).

Changes between version 2.2 and 2.2.1

Improvment of the vertical dotted lines drawn by the specifier ":" in the preamble.
Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of siunitx.
Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of "`|`") as `\hdotsfor` does.
Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.
Error message when the user gives an incorrect value for `last-row`.
A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol "`:`" (in the preamble of the array) and `\line` in `code-after`).
The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.
The vertical rules in the matrices (drawn by "`|`") are now compatible with the color fixed by colortbl.
Correction of a bug: it was not possible to use the colon "`:`" in the preamble of an array when `pdflatex` was used with french-babel (because french-babel activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.
The option `columns-width=auto` doesn't need any more a second compilation.
The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).
The previous version of nicematrix was incompatible with a recent version of expl3 (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange[74], optimization of Tikz externalization is disabled in the environments of nicematrix when the class standalone or the package standalone is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

Changes between version 3.5 and 3.6

LaTeX counters `iRow` and `jCol` available in the cells of the array.
Addition of `\normalbaselines` before the construction of the array: in environments like `{align}` of amsmath the value of `\baselineskip` is changed and if the options `first-row` and `last-row` were used in an environment of nicematrix, the position of the delimiters was wrong.
A warning is written in the `.log` file if an obsolete environment is used.
There is no longer artificial errors `Duplicate~name` in the environments of amsmath.

Changes between version 3.6 and 3.7

The four "corners" of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.
New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programmation for the command `\Block` when the block has only one row. With this programmation, 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`.

[74] cf. `tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize`

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.
Options `vlines`, `hlines` and `hvlines`.
Option `baseline` pour `{NiceArray}` (not for the other environments).
The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell i-j, the name is i-j`-block` and, if the creation of the "medium nodes" is required, a node i-j`-block-medium` is created.
If the user tries to use more columns than allowed by its environment, an error is raised by nicematrix (instead of a low-level error).
The package must be loaded with the option `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the "last row".
The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customisation of the dotted lines.
In the environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, etc.), it's possible to use the options `l` (=`L`) or `r` (=`R`) to specify the type of the columns.
The starred versions of the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots` and `\Iddots` are deprecated since the version 3.1 of nicematrix. Now, one should load nicematrix with the option `starred-commands` to avoid an error at the compilation.
The code of nicematrix no longer uses Tikz but only PGF. By default, Tikz is *not* loaded by nicematrix.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on `stackoverflow`).
Better error messages when the user uses `&` or `\\` when `light-syntax` is in force.

Changes between versions 3.14 and 3.15

It's possible to put labels on the dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and the command `\line` in the `code-after` with the tokens `_` and `^`.
The option `baseline` is now available in all the environments of nicematrix. Before, it was available only in `{NiceArray}`.
New keyword `\CodeAfter` (in the environments of nicematrix).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`
Commands to color cells, rows and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`
New command `\diagbox`
The key `hvline` don't draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It's now possible to write `\begin{pNiceMatrix}a&b\\c&d\end{pNiceMatrix}^2` with the expected result.

Changes between versions 4.2 and 4.3

The horizontal centering of the content of a `\Block` is correct even when an instruction such as `!{\qquad}` is used in the preamble of the array.
It's now possible to use the command `\Block` in the "last row".

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.
It's now possible to use the command `\diagbox` in a `\Block`.
Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).
Environment `{NiceTabular*}`
Command `\Vdotsfor` similar to `\Hdotsfor`
The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.
Key `respect-blocks` for `\rowcolors` (with a *s*) in the `code-before`.
The variable `\g_nicematrix_code_before_tl` is now public.
The key `baseline` may take in as value an expression of the form *line-i* to align the `\hline` in the row *i*.
The key `hvlines-except-corners` may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys `c`, `r` and `l` for the command `\Block`.
It's possible to use the key `draw-first` with `\Ddots` and `\Iddots` to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key `tabularnote`.
Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put `\omit` before `\CodeAfter`.
Correction of a bug: the tabular notes `\tabularnotes` were not composed when present in a block (except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.
New command `\NotEmpty`.

Changes between versions 5.6 and 5.7

New key `delimiters-color`
Keys `fill`, `draw` and `line-width` for the command `\Block`.

Changes between versions 5.7 and 5.8

Keys `cols` and `restart` of the command `\rowcolors` in the `code-before`.
Modification of the behaviour of `\\` in the columns of type `p`, `m` or `b` (for a behaviour similar to the environments of `array`).
Better error messages for the command `\Block`.

Changes between versions 5.8 and 5.9

Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.
New key `cell-space-limits`.

Changes between versions 5.9 and 5.10

New command `\SubMatrix` available in the `\CodeAfter`.
It's possible to provide options (between brackets) to the keyword `\CodeAfter`.
A (non fatal) error is raised when the key `transparent`, which is deprecated, is used.

Changes between versions 5.10 and 5.11

It's now possible, in the `code-before` and in the `\CodeAfter`, to use the syntax `|(i-|j)` for the Tikz node at the intersection of the (potential) horizontal rule number i and the (potential) vertical rule number j.

Changes between versions 5.11 and 5.12

Keywords `\CodeBefore` and `\Body` (alternative syntax to the key `code-before`).
New key `delimiters/max-width`.
New keys `hlines`, `vlines` and `hvlines` for the command `\SubMatrix` in the `\CodeAfter`.
New key `rounded-corners` for the command `\Block`.

Changes between versions 5.12 and 5.13

New command `\arraycolor` in the `\CodeBefore` (with its key `except-corners`).
New key `borders` for the command `\Block`.
New command `\Hline` (for horizontal rules not drawn in the blocks).
The keys `vlines` and `hlines` takes in as value a (comma-separated) list of numbers (for the rules to draw).

Changes between versions 5.13 and 5.14

Nodes of the form `(1.5)`, `(2.5)`, `(3.5)`, etc.
Keys `t` and `b` for the command `\Block`.
Key `corners`.

Changes between versions 5.14 and 5.15

Key hvlines for the command \Block.
The commands provided by nicematrix to color cells, rows and columns don't color the cells which are in the "corners" (when the key corner is used).
It's now possible to specify delimiters for submatrices in the preamble of an environment.
The version 5.15b is compatible with the version 3.0+ of siunitx (previous versions were not).

Changes between versions 5.15 and 5.16

It's now possible to use the cells corresponding to the contents of the nodes (of the form i-j) in the \CodeBefore when the key create-cell-nodes of that \CodeBefore is used. The medium and the large nodes are also available if the corresponding keys are used.

Changes between versions 5.16 and 5.17

The key define-L-C-R (only available at load-time) now raises a (non fatal) error.
Keys L, C and R for the command \Block.
Key hvlines-except-borders.
It's now possible to use a key l, r or c with the command \pAutoNiceMatrix (and the similar ones).

Changes between versions 5.17 and 5.18

New command \RowStyle

Changes between versions 5.18 and 5.19

New key tikz for the command \Block.

Changes between versions 5.19 and 6.0

Columns X and environment {NiceTabularX}.
Command \rowlistcolors available in the \CodeBefore.
In columns with fixed width, the blocks are composed as paragraphs (wrapping of the lines).
The key define-L-C-R has been deleted.

Changes between versions 6.0 and 6.1

Better computation of the widths of the X columns.
Key \color for the command \RowStyle.

Changes between versions 6.1 and 6.2

Better compatibility with the classes revtex4-1 and revtex4-2.
Key vlines-in-sub-matrix.

Changes between versions 6.2 and 6.3

Keys nb-rows, rowcolor and bold for the command \RowStyle
Key name for the command \Block.
Support for the columns V of varwidth.

Changes between versions 6.3 and 6.4

New commands \UnderBrace and \OverBrace in the \CodeAfter.
Correction of a bug of the key baseline (cf. question 623258 on TeX StackExchange).
Correction of a bug with the columns V of varwidth.
Correction of a bug: the use of \hdottedline and : in the preamble of the array (of another letter specified by letter-for-dotted-lines) was incompatible with the key xdots/line-style.

Changes between versions 6.4 and 6.5

Key custom-line in \NiceMatrixOptions.
Key respect-arraystretch.

Changes between version 6.5 and 6.6

Keys tikz and width in custom-line.

Changes between version 6.6 and 6.7

Key color for \OverBrace and \UnderBrace in the \CodeAfter
Key tikz in the key borders of a command \Block

Changes between version 6.7 and 6.8

In the notes of a tabular (with the command \tabularnote), the duplicates are now detected: when several commands \tabularnote are used with the same argument, only one note is created at the end of the tabular (but all the labels are present, of course).

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Contents