The makecell package[*]

Olga Lapko
`Lapko.O@g23.relcom.ru`

2006/06/28

Abstract

This package helps to create common layout for tabular material. The `\thead` command, based on one-column tabular environment, is offered for creation of tabular column heads. This macro allows to support common layout for tabular column heads in whole documentation. Another command, `\makecell`, is offered for creation of multilined tabular cells.

Package also offers: 1) macro `\makegapedcells`, which changes vertical spaces around all cells in tabular, like in tabls package, but uses code of array package. (Macro `\makegapedcells` redefines macro `\@classz` from this package. Macro `\nomakegapedcells` cancels this redefinition.); 2) macros `\multirowhead` and `\multirowcell`, which use `\multirow` macro from multirow package.

1 Tabular Cells and Column Heads

1.1 Building Commands

`\makecell` Macro creates one-column tabular with predefined common settings of alignment, spacing and vertical spaces around (see section 1.2). This will be useful for creation of multilined cells. This macro allows optional alignment settings.

`\makecell[`⟨*vertical or/and horizontal alignment*⟩`]{`⟨*cell text*⟩`}`

For vertical alignment you use `t`, `b`, or `c`—this letters you usually put in optional argument of `tabular` or `array` environments. For horizontal alignment you may use alignment settings like `r`, `l`, or `c`, or more complex, like `{p{3cm}}`. Since this package loads array package, you may use such alignment settings like `{>{\parindent1cm}p{3cm}}`.

```
\begin{tabular}{|c|c|}
\hline
Cell text & 28--31\\
\hline
```

[*]This file has version number v0.1c, last revised 2006/06/28.

```
\makecell{Multilined \\ cell text} & 28--31\\
\hline
\makecell[l]{Left aligned \\ cell text} & 37--43\\
\hline
\makecell*[r]{Right aligned \\ cell text} & 37--43\\
\hline
\makecell[b]{Bottom aligned \\ cell text} & 52--58\\
\hline
\makecell*[{{p{3cm}}}]{Cell long text with predefined width} & 52--58\\
\hline
\makecell[{{>{\parindent1em}p{3cm}}}]{Cell long...} & 52--58\\
\hline
\end{tabular}
```

Table 1. Example of multilined cells

Cell text	28–31
Multilined cell text	28–31
Left aligned cell text	37–43
Right aligned cell text	37–43
Bottom aligned cell text	52–58
Cell long text with predefined width	52–58
Cell long text with predefined width	52–58

Starred form of command, `\makecell*`, creates vertical `\jot` spaces around.

Note. When you define column alignment like `p{3cm}` in optional argument of `\makecell` (or `\thead`, see below), please follow these rules: 1) if vertical alignment defined, write column alignment in group, e.g. `[c{p{3cm}}]`; 2) if it is absent, write column alignment in double group—`[{{p{3cm}}}]`, or add empty group—`[{}{p{3cm}}]`. Be also careful with vertical alignment when you define column alignment as paragraph block: e.g., use `{{b{3cm}}}` for bottom alignment (and `{{m{3cm}}}` for centered vertical alignment).

`\thead` Macro creates one-column `tabular` for column heads with predefined common settings (see table 2). This macro uses common layout for column heads: font, alignment, spacing, and vertical spaces around (see section 1.2).

```
\renewcommand\theadset{\def\arraystretch{.85}}%
\begin{tabular}{|l|c|}
\hline
\thead{First column head}&
  \thead{Second \\multlined \\ column head}\\
\hline
Left column text & 28--31\\
\hline
\end{tabular}
```

Table 2. Example of column heads

First column head	Second multlined column head
Long left column text	28–31

Starred form of command, `\thead*`, creates vertical `\jot` spaces around.

`\rothead` Creates table heads rotated by 90° counterclockwise. Macro uses the same font and spacing settings as previous one, but column alignment changed to `p{\rotheadsize}` with `\raggedright` justification: in this case left side of all text lines "lies" on one base line.

`\rotheadsize` This parameter defines the width of rotated tabular heads. You may define that like:

```
\setlength\rotheadsize{3cm}
```

or

`\settowidth\rotheadsize{\theadfont` ⟨*Widest head text*⟩`}`

like in following example:

```
\settowidth\rotheadsize{\theadfont Second multilined}
\begin{tabular}{|l|c|}
\hline
\thead{First column head}&
  \rothead{Second multilined \\ column head}\\
\hline
Left column text & 28--31\\
\hline
\end{tabular}
```

Table 3. Example of rotated column heads

First column head	Second multilined column head
Long left column text	28–31

1.2 Settings For Tabular Cells

This section describes macros, which make layout tuning for multilined cells, created by `\makecell` macro (and also `\multirowcell` and `\rotcell` macros). The `\cellset` macro also is used by `\thead` (`\rothead`, `\multirowtead`) macro.

`\cellset` Spacing settings for cells. Here you could use commands like:

```
\renewcommand\cellset{\renewcommand\arraytretch{1}%
    \setlength\extrarowheight{0pt}}
```

as was defined in current package. These settings used by both `\makecell` and `\thead` (`\rothead`, `\multirowtead`) commands.

`\cellalign` Default align for cells. Package offers vertical and horizontal centering alignment, it defined like:

```
\renewcommand\cellalign{cc}
```

Define vertical spaces around `\makecell`, using `\gape` command if necessary. It defined like:

```
\renewcommand\cellgape{}
```

You may define this command like

```
\renewcommand\cellgape{\Gape[1pt]}
```

or

```
\renewcommand\cellgape{\gape[t]}
```

(See also section 2 about `\gape` command.)

`\cellrotangle` The angle for rotated cells and column heads. The default value 90 (counterclockwise). This value definition is used by both `\rotcell` and `\rothead` macros.

1.3 Settings For Column Heads

This section describes macros, which make layout tuning for tabular column heads, created by `\thead` (`\rothead`, `\multirowtead`) macro.

`\theadfont` Sets a special font for column heads. It could be smaller size

```
\renewcommand\theadfont{\foonotesize}
```

as was defined in current package (here we suppose that `\small` command used for tabular contents itself). Next example defines italic shape

```
\renewcommand\theadfont{\itshape}
```

`\theadset` Spacing settings for column heads. Here you could use commands like:

```
\renewcommand\theadset{\renewcommand\arraytretch{1}%
    \setlength\extrarowheight{0pt}}
```

`\theadalign` Default align for tabular column heads. Here also offered centering alignment:

```
\renewcommand\theadalign{cc}
```

`\theadgape` Define vertical spaces around, using `\gape` command if necessary. It defined like:

```
\renewcommand\theadgape{\gape}
```

`\rotheadgape` Analogous definition for rotated column heads. Default is absent:

```
\renewcommand\rotheadgape{}
```

2 Changing of Height and Depth of Boxes

Sometimes `tabular` or `array` cells, or some elements in text need a height/depth correction. The `\raisebox` command could help for it, but usage of that macro in these cases, especially inside math, is rather complex. Current package offers the `\gape` macro, which usage is similar to `\smash` macro. The `\gape` macro allows to change height and/or depth of included box with necessary dimension.

`\gape` This macro changes included box by `\jot` value (usually 3 pt). It is defined with optional and mandatory arguments, like `\smash` macro, which (re)defined by `amsmath` package. Optional argument sets change of height only (`t`) or depth only (`b`). Mandatory argument includes text.

`\gape[`⟨*t or b*⟩`]{`⟨*text*⟩`}`

Examples of usage:

`\gape{text}` `\gape[t]{text}` `\gape[b]{text}`

`\Gape` Another way of height/depth modification. This macro allows different correction for height and depth of box:

`\Gape[`⟨*height corr*⟩`][`⟨*depth corr*⟩`]{`⟨*text*⟩`}`

If both arguments absent, `\Gape` command works like `\gape{`⟨*text*⟩`}`, in other words, command uses `\jot` as correction value for height and depth of box.

If only one optional argument exists, `\Gape` command uses value from this argument for both height and depth box corrections.

`\Gape{text}` `\Gape[\jot]{text}`

`\Gape[6pt]{text}` `\Gape[6pt][-2pt]{text}`

`\bottopstrut` `\topstrut` `\botstrut` These three macros modify standard `\strut` by `\jot` value: `\bottopstrut` changes both height and depth; `\topstrut` changes only height; `\botstrut` changes only depth. These commands could be useful, for example, in first and last table rows.

Note. If you use bigstrut package note that these macros duplicate `\bigstrut`, `\bigstrut[t]`, and `\bigstrut[b]` commands consequently. Please note that value, which increases strut in `\topstrut` etc. equals to `\jot`, but `\bigstrut` and others use a special dimension `\bigstrutjot`.

3 How to Change Vertical Spaces Around in Whole Table

This section describes macros which try to emulate one of possibilities of tabls package: to get necessary vertical spacing around cells.

`\setcellgapes` Sets the parameters for vertical spaces. It looks like `\gape*` command without `{⟨text⟩}` argument:

```
\setcellgapes[⟨t or b⟩]{⟨value⟩}
```

The next examples with array and tabular use following settings:

```
\setcellgapes{5pt}
```

You may also try to load negative values if you wish. This macro you may put in the preamble as common settings.

`\makegapedcells`
`\nomakegapedcells` The first macro switches on vertical spacing settings. The second cancels first one.

The `\makegapedcells` macro temporarily redefines macro `\@classz` of array package, so use this mechanism carefully. Load `\makegapedcells` inside group or inside environment (see table 4):

```
\begin{table}[h]
\makegapedcells
...
\end{table}
```

Please note that space defined in `\setcellgapes` and space which creates `\gape` mechanism in commands for tabular cells (usually `\thead` or `\makecell*`) are summarized.

4 Multirow Table Heads and Cells

The next examples show usage of macros which use `\multirow` command from multirow package.

At first goes short repetition of arguments of `\multirow` macro itself:

```
\multirow{⟨nrow⟩}[⟨njot⟩]{⟨width⟩}[⟨vmove⟩]{⟨contents⟩}
```

`{⟨nrow⟩}` sets number of rows (i.e. text lines); `[⟨njot⟩]` is mainly used if you've used bigstrut package: it makes additional tuning of vertical position (see comments in multirow package); `{⟨width⟩}` defines width of contents, the `*` sign used

Table 4. Example of multilined cells with additional vertical spaces

Cell text	28–31
Multilined cell text	28–31
Left aligned cell text	37–43
Right aligned cell text	37–43
Bottom aligned cell text	52–58
Cell long text with predefined width	52–58
Cell long text with predefined width	52–58

to indicate that the text argument's natural width is to be used; `[`⟨*vmove*⟩`]` is a length used for fine tuning: the text will be raised (or lowered, if ⟨*vmove*⟩ is negative) by that length; `{`⟨*contents*⟩`}` includes "`\multirow`'ed" text.

`\multirowcell`
`\multirowthead`

These two macros use following arguments (example uses `\multirowcell` command):

`\multirowcell{`⟨*nrow*⟩`}[`⟨*vmove*⟩`][`⟨*hor alignment*⟩`]{`⟨*contents*⟩`}`

in these macros were skipped `[`⟨*njot*⟩`]` and `{`⟨*width*⟩`}`. Instead of tuning optional argument `[`⟨*njot*⟩`]` for vertical correction used `[`⟨*vmove*⟩`]` optional argument. For the `{`⟨*width*⟩`}` argument both `\multirowcell` and `\multirowthead` macros use natural width of contents (i.e. the `*` argument used).

First example (table 5) with "`\multirow`'ed" column heads and cells:

```
\renewcommand\theadset{\def\arraystretch{.85}}%
\begin{tabular}{|l|c|c|}
 \multirowthead{4}{First ...}&
 \multicolumn{2}{c|}{\thead{Multicolumn head}}\\               \cline{2-3}
   & \thead{Second ...} & \thead{Third ...}\\                   \hline
 Cell text & A &\multirowcell{3}{28--31}\\                      \cline{1-2}
 \makecell{Multilined\\Cell text} & B& \\                         \hline
 \makecell[l]{Left ...} & C & \multirowcell{4}[1ex][l]{37--43}\\ \cline{1-2}
 \makecell[r]{Right ...} & D & \\                                \hline
 \makecell[b]{Bottom ...} & E & \multirowcell{5}[1ex][r]{37--43\\52--58}\\
\cline{1-2}
\makecell[{{p{5cm}}}]{Cell ...} & F & \\                        \cline{1-2}
\makecell[{{>{\parindent1em}p{5cm}}}]{Cell ...} & G & \\        \hline
\end{tabular}
```

Second example (table 6) with "multirow'ed" column heads and cells uses `\makegapedcells` command. The `\theadgape` command does nothing:

```
\makegapedcells
\renewcommand\theadset{\def\arraystretch{.85}}%
\renewcommand\theadgape{}
...
```

The last example (table 7) uses `tabularx` environment with `\hsize` in the width argument.

```
\makegapedcells
\renewcommand\theadset{\def\arraystretch{.85}}%
\renewcommand\theadgape{}
\begin{tabularx}\hsize{|X|c|c|}
...
\cline{1-2}
\makecell[{{p{\hsize}}}]{Cell ...} & F & \\
\cline{1-2}
\makecell[{{>{\parindent1em}p{\hsize}}}]{Cell ...} & G & \\
\hline
\end{tabularx}
```

As you may see the `\makecell`'s in last two rows defined as

```
\makecell[{{p{\hsize}}}]{...}
```

and

```
\makecell[{{>{\parindent1em}p{\hsize}}}]{...}
```

consequently.

4.1 Multirow Table Heads and Cells: Second Variant

Another, simplified, variant of multirow cell: use `\makecell` and `\thead` commands, and set `\\` with negative space at the end, for example

```
\thead{First Column head\\[-5ex]}
```

cells, which stay in one "multi row" will have the same value of this negative space, in spite of different number of lines in their contents.

Table 5. Example of "`\multirow`'ed" cells

First Column head	Multicolumn head	
	Second multlined column head	Third column head
Cell text	A	28–31
Multilined Cell text	B	
Left aligned cell text	C	37–43
Right aligned cell text	D	
Bottom aligned cell text	E	37–43 52–58
Cell long long long long text with predefined width	F	
Cell long long long long text with predefined width	G	

Table 6. Example of "`\multirow`'ed" cells and additional vertical spaces

First Column head	Multicolumn head	
	Second multlined column head	Third column head
Cell text	A	28–31
Multilined Cell text	B	
Left aligned cell text	C	37–43
Right aligned cell text	D	
Bottom aligned cell text	E	37–43 52–58
Cell long long long long text with predefined width	F	
Cell long long long long text with predefined width	G	

Table 7. Example of `tabularx` environment

First Column head	Multicolumn head	
	Second multlined column head	Third column head
Cell text	A	28–31
Multilined Cell text	B	
Left aligned cell text	C	37–43
Right aligned cell text	D	
Bottom aligned cell text	E	37–43 52–58
Cell long long long long long long text with predefined width	F	
Cell long long long long long long text with predefined width	G	

Table 8. Examples of filling of cells

1	2	3	4	5	6
			4	5	6
(a)	(b)	(c)	(d)	(e)	(f)
column I	column II	column III	column IV	column V	column VI

5 Numbered Lines in Tabulars

The three commands `\eline`, `\nline`, `\rnline` allow to skip:

`\eline{`⟨*number of cells*⟩`}`

and numbering (`\nline`) a few/all sells in the row:

`\nline[`⟨*numbering type*⟩`][`⟨*start number*⟩`]{`⟨*number of cells*⟩`}`

Command `\rnline` does the same as `\nline`, but allows numbering by Russian letters (it redefines LaTeX's `\Alph` and `\alph` with `\Asbuk` and `\asbuk` consequently). (see table 8)

```
\begin{tabular}{|*{12}{c|}}
\hline
\eline{6}                    \\ \hline
\nline{6}                    \\ \hline
\eline{3} & \nline[1][4]{3} \\ \hline
\nline[(a)]{6}               \\ \hline
\nline[column I]{6}          \\ \hline
\end{tabular}
```

6 Code of package

6.1 Multilined cells

First goes request of array package.

```
\RequirePackage{array}
```

\makecell The definition of command for multilined cells. At first defined \gape stuff. Non-star form loads special setting for vertical space around (if it used). Star form always creates additional vertical \jot-spaces.

```
\newcommand\makecell{\@ifstar{\let\tabg@pe\gape\makecell@}%
                              {\let\tabg@pe\cellgape\makecell@}}
```

Next macro loads vertical and horizontal common alignment for cells and loads redefined spacing parameters \arraystretch and \extrarowheight if these parameters were redefined.

```
\newcommand\makecell@{\def\t@bset{\cellset}%
   \let\mcell@align\cellalign
   \@ifnextchar[\mcell@tabular
     {\expandafter\mcell@@tabular\cellalign\@nil}}
```

\thead The macro for tabular column heads. At first defined \gape stuff. Non-star from loads special setting for vertical space around (if it used). Star form always creates additional vertical \jot-spaces.

```
\newcommand\thead{\@ifstar{\let\tabg@pe\gape\thead@}%
                          {\let\tabg@pe\theadgape\thead@}}
```

Next macro loads vertical and horizontal common alignment for column heads and loads redefined spacing parameters \arraystretch and \extrarowheight if these parameters were redefined. (First go settings for cells, as for \makecell, then special settings for column heads.)

For column heads also loaded font settings.

```
\newcommand\thead@{\def\t@bset{\cellset\theadfont\theadset}%
   \let\mcell@align\theadalign
   \@ifnextchar[\mcell@tabular
     {\expandafter\mcell@@tabular\theadalign\@nil}}
```

\rotheadsize The width dimension for rotated cells.

```
\@ifdefinable\rotheadsize{\newdimen\rotheadsize}
```

\rotcell The macro for rotated cell. If no rotating package loaded this macro works like \makecell.

```
\newcommand\rotcell{\@ifundefined{turn}%
  {\PackageWarning{makecell}%
     {\string\rotcell\space needs rotating package}%
  \let\tabg@pe\empty\let\t@bset\cellset\makecell@}
  {\@ifnextchar[{\@rotcell}{\@@rotcell}}}
```

For rotated cell default column setting is similar to `p{\rotheadsize}` (plus some additional justification settings)

```
\@ifdefinable\@rotcell{}
\def\@rotcell[#1]#2{\makecell{\\[-.65\normalbaselineskip]
  \turn{\cellrotangle}\makecell[#1]{#2}\endturn}}
\newcommand\@@rotcell[1]{\makecell{\\[-.65\normalbaselineskip]
  \turn{\cellrotangle}\makecell[c{>{\rightskip0explus
    \rotheadsize\hyphenpenalty0\pretolerance-1%
    \noindent\hskip\z@}p{\rotheadsize}
    }]{#1}\endturn}}
```

`\rothead` The macro for rotated tabular column heads. If no rotating package loaded this macro works like `\thead`.

```
\newcommand\rothead{\@ifundefined{turn}%
  {\PackageWarning{makecell}{\string\rothead\space
    needs rotating package}%
   \let\tabg@pe\theadgape
   \def\t@bset{\cellset\theadfont\theadset}\thead@}%
  {\let\theadgape\rotheadgape
   \@ifnextchar[{\@rothead}{\@@rothead}}}
```

For rotated column head default column setting is similar to `p{\rotheadsize}` (plus some additional justification settings)

```
\@ifdefinable\@rothead{}
\def\@rothead[#1]#2{\thead{\\[-.65\normalbaselineskip]
  \turn{\cellrotangle}\thead[#1]{#2@{}}\endturn}}
\newcommand\@@rothead[1]{\thead{\\[-.65\normalbaselineskip]
  \turn{\cellrotangle}\thead[c{>{\rightskip0explus
    \rotheadsize\hyphenpenalty0\pretolerance-1%
    \noindent\hskip\z@}p{\rotheadsize}
    @{}}]{#1}\endturn}}
```

`\multirowcell` The macro for multirow cells. If no multirow package loaded this macro works like `\makecell`.

```
\newcommand\multirowcell{\@ifundefined{multirow}%
  {\PackageWarning{makecell}{\string\multirowcell\space
   needs multirow package}}%
  {\let\mcell@multirow\multirow}\mcell@mrowcell@}
```

These macros define settings for `\multirow` arguments.

```
\newcommand\mcell@mrowcell@[1]{\@ifnextchar
  [{\mcell@mrowcell@@{#1}}{\mcell@mrowcell@@{#1}[0pt]}}
\@ifdefinable\mcell@mrowcell@@{}
\def\mcell@mrowcell@@#1[#2]{\edef\mcell@nrows{#1}\edef\mcell@fixup{#2}%
   \let\tabg@pe\cellgape\makecell@}
```

`\multirowthead` The macro for multirow column heads. If no multirow package loaded this macro works like `\thead`.

```
\newcommand\multirowthead{\@ifundefined{multirow}%
```

```
 {\PackageWarning{makecell}{\string\multirowthead\space
  needs multirow package}}%
 {\let\mcell@multirow\multirow}\mcell@mrowhead@}
```

These macros define settings for `\multirow` arguments.

```
\newcommand\mcell@mrowhead@[1]{\@ifnextchar
  [{\mcell@mrowhead@@{#1}}{\mcell@mrowhead@@{#1}[0pt]}}
\@ifdefinable\mcell@mrowhead@@{}
\def\mcell@mrowhead@@#1[#2]{\edef\mcell@nrows{#1}\edef\mcell@fixup{#2}%
   \let\tabg@pe\theadgape\thead@}
```

`\mcell@multirow` By default `\mcell@multirow` macro gobbles `\multirow`'s arguments.

```
\@ifdefinable\mcell@multirow{}
\def\mcell@multirow#1#2[#3]{}%
```

Definitions for horizontal and vertical alignments, which use by `tabular` and `array` environments.

For `l`, `r`, `t`, and `b` alignments commands set `c`-argument as vertical or horizontal centering alignment if necessary. For `l` and `r` alignments also redefined alignment settings for `\makecell` (`\thead`) blocks.

```
\newcommand\mcell@l{\def\mcell@ii{l}\let\mcell@c\mcell@ic
  \global\let\mcell@left\empty}
\newcommand\mcell@r{\def\mcell@ii{r}\let\mcell@c\mcell@ic
  \global\let\mcell@right\empty}
\newcommand\mcell@t{\def\mcell@i{t}\let\mcell@c\mcell@iic}
\newcommand\mcell@b{\def\mcell@i{b}\let\mcell@c\mcell@iic}
\newcommand\mcell@{}
```

If alone `c`-argument loaded it is used for horizontal alignment.

```
\newcommand\mcell@c{\def\mcell@ii{c}}
\newcommand\mcell@ic{\def\mcell@i{c}}
\newcommand\mcell@iic{\def\mcell@ii{c}}
```

Default vertical and horizontal alignment is centered.

```
\newcommand\mcell@i{c}
\newcommand\mcell@ii{c}
```

Default horizontal alignment of `\makecell` (`\thead`) blocks is centered.

```
\@ifdefinable\mcell@left{\let\mcell@left\hfill}
\@ifdefinable\mcell@right{\let\mcell@right\hfill}
```

`\mcell@tabular` `\mcell@@tabular` `\mcell@@@tabular`

The core macros for tabular building.

Next few macros for sorting of `\makecell` (`\thead`) arguments.

```
\@ifdefinable\mcell@tabular{}\@ifdefinable\mcell@@tabular{}
\@ifdefinable\mcell@@@tabular{}
\def\mcell@tabular[#1]#2{\mcell@@tabular#1\@nil{#2}}
```

The code for this macro borrowed from caption 3.x package (AS).

```
\newcommand\mcell@ifinlist[2]{%
  \let\next\@secondoftwo
  \edef\mcell@tmp{#1}%
  \@for\mcell@Tmp:={#2}\do{%
    \ifx\mcell@tmp\mcell@Tmp
      \let\next\@firstoftwo
    \fi}\next}
```

The `\mcell@@tabular` macro at first calls `\mcell@setalign` macro for sorting of alignment arguments, then calls `\mcell@@@tabular` macro, which created tabular cell or column head.

```
\def\mcell@@tabular#1#2\@nil#3{%
  \expandafter\mcell@setalign\mcell@align\@nil
  \mcell@setalign{#1}{#2}\@nil
  \expandafter\mcell@@@tabular\expandafter\mcell@i\mcell@ii\@nil{#3}}
```

`\mcell@setalign` This macro sorts arguments for vertical and horizontal alignment.

First argument has second check at the end of macro for the case if it is `c`-argument.

```
\@ifdefinable\mcell@setalign{}
\def\mcell@setalign#1#2\@nil{\def\@tempa{#1}\def\@tempc{c}%
```

Restore default alignment for `\makecell` and `\thead` blocks.

```
  \global\let\mcell@left\hfill\global\let\mcell@right\hfill
```

If in optional argument appears alone `c`-argument it defines horizontal centering only.

```
  \def\mcell@c{\def\mcell@ii{c}}%
  \mcell@ifinlist{#1}{l,r,t,b,c,}{\@nameuse{mcell@#1}}%
```

If argument is not `l`, `r`, `c`, `t`, or `b` it could define horizontal alignment only.

```
      {\def\mcell@ii{#1}\let\mcell@c\mcell@ic
       \let\mcell@left\empty\let\mcell@right\empty}%
  \mcell@ifinlist{#2}{l,r,t,b,c,}{\@nameuse{mcell@#2}}%
```

If argument is not `l`, `r`, `c`, `t`, or `b` it could define horizontal alignment only.

```
      {\def\mcell@ii{#2}\let\mcell@c\mcell@ic
       \let\mcell@left\empty\let\mcell@right\empty}%
```

Here goes repeated check for first argument, if it is `c`-argument we call `\mcell@c` command, which can be now redefined.

```
  \ifx\@tempa\@tempc\mcell@c\fi
}
```

This macro builds tabular itself. First (and last) go commands which align `\makecell` and `\thead` blocks like `l`, `r`, or `c` (if they loaded). Then goes check whether math mode exists. The `\mcell@multirow` emulation macro transforms to `\multirow` when necessary.

```
\def\mcell@@@tabular#1#2\@nil#3{%\mcell@mstyle
  \ifdim\parindent<\z@\leavevmode\else\noindent\fi
  \null\mcell@left
      \ifmmode
         \mcell@multirow\mcell@nrows*[\mcell@fixup]{\tabg@pe
          {\hbox{\t@bset$\array[#1]{@{}#2@{}}#3\endarray$}}}%
```

```
     \else
        \mcell@multirow\mcell@nrows*[\mcell@fixup]{\tabg@pe
         {\hbox{\t@bset\tabular[#1]{@{}#2@{}}#3\endtabular}}}%
     \fi\mcell@right\null}
```

\cellset \cellgape \cellalign \cellrotangle \theadfont \theadset \theadgape \rotheadgape \theadalign

The layout macros for tabular building settings.

Spacing settings for tabular spacing inside cells (like `\arraystretch` or `\extrarowheight`).

```
\newcommand\cellset{\def\arraystretch{1}\extrarowheight\z@
   \nomakegapedcells}
```

Vertical space around cells (created by `\gape` stuff).

```
\newcommand\cellgape{}
```

Vertical and horizontal alignment of cell text.

```
\newcommand\cellalign{cc}
```

Angle for rotated column heads and cells.

```
\newcommand\cellrotangle{90}
```

Font for column heads

```
\newcommand\theadfont{\footnotesize}
```

Special spacing settings for tabular spacing in column heads (like `\arraystretch` or/and `\extrarowheight`).

```
\newcommand\theadset{}
```

Vertical space around column heads (created by `\gape` stuff).

```
\newcommand\theadgape{\gape}
```

Vertical space around rotated column heads.

```
\newcommand\rotheadgape{}
```

Vertical and horizontal alignment of column head text.

```
\newcommand\theadalign{cc}
```

6.2 Gape commands

\gape \setcellgapes

The macro itself. It uses analogous to `\smash` macro from `amsmath` package. Starred form has additional mandatory argument for value of `\gape`.

```
\newcommand\gape{\@ifnextchar[\@gape{\@gape[tb]}}
```

The `\setcellgapes` defines settings used by `\makegapedcells` command. First goes check for optional argument.

```
\newcommand\setcellgapes{\@ifnextchar[%]
  {\mcell@setgapes{MB}}{\mcell@setgapes{MB}[tb]}}
```

Then body of settings.

```
\@ifdefinable\@setcellgapes{}
\def\mcell@setgapes#1[#2]#3{\expandafter\let\csname
  mcell@#1@\expandafter\endcsname\csname mcell@mb@#2\endcsname
 \@namedef{mcell@#1jot}{#3}}
```

The macros which count advanced height and depth of boxes.

```
\newcommand\mcell@mb@t[2]{\@tempdima\ht#1\advance\@tempdima#2%
  \ht#1\@tempdima}
\newcommand\mcell@mb@b[2]{\@tempdimb\dp#1\advance\@tempdimb#2%
  \dp#1\@tempdimb}
\newcommand\mcell@mb@tb[2]{\mcell@mb@t{#1}{#2}\mcell@mb@b{#1}{#2}}
```

The body of `\gape` macros.

```
\@ifdefinable\@gape{}\@ifdefinable\@@gape{}
\def\@gape[#1]{\mcell@setgapes{mb}[#1]{\jot}\@@gape}
\def\@@gape{%
  \ifmmode \expandafter\mathpalette\expandafter\mathg@pe
  \else \expandafter\makeg@pe
  \fi}
```

`\makeg@pe` The macros which put box with necessary parameters in text and math mode.

```
\newcommand\makeg@pe[1]{\setbox\z@
  \hbox{\color@begingroup#1\color@endgroup}\mcell@mb@\z@\mcell@mbjot\box\z@}
\newcommand\mathg@pe[2]{\setbox\z@
  \hbox{$\m@th#1{#2}$}\mcell@mb@\z@\mcell@mbjot\box\z@}
```

`\Gape` The macros which put box with necessary parameters in text and math mode.

```
\newcommand\Gape{\@ifnextchar[\@Gape{\@Gape[\jot]}}
\@ifdefinable\@Gape{}\@ifdefinable\@@Gape{}
\def\@Gape[#1]{\@ifnextchar[{\@@Gape[#1]}{\@@Gape[#1][#1]}}
\def\@@Gape[#1][#2]{\def\depth{\dp\z@}\def\height{\ht\z@}%
  \edef\mcell@mb@##1##2{%
    \@tempdima\ht\z@\advance\@tempdima#1\ht\z@\@tempdima
    \@tempdimb\dp\z@\advance\@tempdimb#2\dp\z@\@tempdimb}%
    \@@gape}
```

`\topstrut` `\botstrut` `\bottopstrut` The macros abbreviations for `\strut` which changed by value of `\jot`. First enlarges both depth and height.

```
\newcommand\bottopstrut{\gape{\strut}}
```

Second enlarges only height.

```
\newcommand\topstrut{\gape[t]{\strut}}
```

Third enlarges only depth.

```
\newcommand\botstrut{\gape[b]{\strut}}
```

6.3 Modification of command from array package

`\makegapedcells` `\nomakegapedcells` At first is saved `\@classz` macro.

```
\@ifdefinable\mcell@oriclassz{\let\mcell@oriclassz\@classz}
```

This macros redefine and restore the `\@classz` macro from array package.

```
\newcommand\makegapedcells{\let\@classz\mcell@classz}
\newcommand\nomakegapedcells{\let\@classz\mcell@oriclassz}
```

`\mcell@agape` Following macro creates tabular/array cells with changed vertical spaces.

```
\newcommand\mcell@agape[1]{\setbox\z@\hbox{#1}\mcell@MB@\z@\mcell@MBjot
   \null\mcell@left\box\z@\mcell@right\null}
```

`\mcell@classz` Redefined `\@classz` macro from array package.

```
\newcommand\mcell@classz{\@classx
   \@tempcnta \count@
   \prepnext@tok
   \@addtopreamble{%\mcell@mstyle
      \ifcase\@chnum
         \hfil
         \mcell@agape{\d@llarbegin\insert@column\d@llarend}\hfil \or
         \hskip1sp
         \mcell@agape{\d@llarbegin\insert@column\d@llarend}\hfil \or
         \hfil\hskip1sp
         \mcell@agape{\d@llarbegin \insert@column\d@llarend}\or
         $\mcell@agape{\vcenter
         \@startpbox{\@nextchar}\insert@column\@endpbox}$\or
         \mcell@agape{\vtop
         \@startpbox{\@nextchar}\insert@column\@endpbox}\or
         \mcell@agape{\vbox
         \@startpbox{\@nextchar}\insert@column\@endpbox}%
      \fi
      \global\let\mcell@left\relax\global\let\mcell@right\relax
    }\prepnext@tok}
```

6.4 Rows of skipped and numbered cells

`\eline` The row of empty cells.

```
\newcommand\eline[1]{\count@ #1%
  \advance\count@\m@ne
   \loop \@temptokena\expandafter{\the\@temptokena&}%
    \advance\count@\m@ne \ifnum\count@>\z@\repeat
     \the\@temptokena\ignorespaces}
```

`\rnline` `\nline` The rows of numbered cells. The `\rnline` command replaces `\Alph` and `\alph` counter by `\Asbuk` and `\asbuk` consequently.

```
\newcommand\rnline{\gdef
   \TeXr@rus{\let\@Alph\@Asbuk\let\@alph\@asbuk}\@nline}
\newcommand\nline{\gdef\TeXr@rus{}\@nline}
\newcommand\@nline{\@ifnextchar[%]
   {\@@nline}{\@@nline[1]}}
\@ifdefinable\@@nline{}
\def\@@nline[#1]{\@ifnextchar[%]
   {\@@@nline[#1]}{\@@@nline[#1][1]}}
\@ifdefinable\@@@nline{}
\def\@@@nline[#1][#2]#3{\count@ #3%
  \expandafter\TeXr@loop\@gobble{}#1\@@@
  \xdef\Num{\the\TeXr@lab}%
\@tempcnta#2\relax%
\expandafter\@temptokena\expandafter{\Num
  \global\advance\@tempcnta\@ne}%
\advance\count@\m@ne
\loop\@temptokena\expandafter{\the\@temptokena&
      \Num \global\advance\@tempcnta\@ne}%
  \advance\count@\m@ne \ifnum\count@>\z@ \repeat
   \the\@temptokena\ignorespaces}
```

[Borrowed code stuff and explanation from enumerate/paralist packages just with changes of command names.]

Internal token register used to build up the label command from the optional argument.

```
\newtoks\TeXr@lab
```

This just expands to a '?'. `\ref` will produce this, if no counter is printed.

```
\def\TeXr@qmark{?}
```

The next four macros build up the command that will print the item label. They each gobble one token or group from the optional argument, and add corresponding tokens to the register `\@enLab`. They each end with a call to `\@enloop`, which starts the processing of the next token.

`\TeXr@label` Add the counter to the label. `#2` will be one of the 'special' tokens `A a I i 1`, and is thrown away. `#1` will be a command like `\Roman`.

```
\def\TeXr@label#1#2{%
  \xdef\TeXr@the{\noexpand#1\@tempcnta}%
  \TeXr@lab\expandafter{\the\TeXr@lab\TeXr@rus\TeXr@the}%
  \advance\@tempcnta1
  \TeXr@loop}
```

The only foreign command in this stuff. It indicates whether the list has numeration by Russian letters.

```
\def\TeXr@rus{}
```

`\TeXr@space` `\TeXr@sp@ce` Add a space to the label. The tricky bit is to gobble the space token, as you can not do this with a macro argument.

```
\def\TeXr@space{\afterassignment\TeXr@sp@ce\let\@tempa= }
\def\TeXr@sp@ce{\TeXr@lab\expandafter{\the\TeXr@lab\space}\TeXr@loop}
```

`\TeXr@group` Add a `{ }` group to the label.

```
\def\TeXr@group#1{\TeXr@lab\expandafter{\the\TeXr@lab{#1}}\TeXr@loop}
```

`\TeXr@other` Add anything else to the label

```
\def\TeXr@other#1{\TeXr@lab\expandafter{\the\TeXr@lab#1}\TeXr@loop}
```

`\TeXr@loop` `\TeXr@loop@` The body of the main loop. Eating tokens this way instead of using `\@tfor` lets you see spaces and **all** braces. `\@tfor` would treat `a` and `{a}` as special, but not `{{a}}`.

```
\def\TeXr@loop{\futurelet\TeXr@temp\TeXr@loop@}
```

```
\def\TeXr@loop@{%
  \ifx A\TeXr@temp         \def\@tempa{\TeXr@label\@Alph  }\else
  \ifx a\TeXr@temp         \def\@tempa{\TeXr@label\@alph  }\else
  \ifx i\TeXr@temp         \def\@tempa{\TeXr@label\@roman }\else
  \ifx I\TeXr@temp         \def\@tempa{\TeXr@label\@Roman }\else
  \ifx 1\TeXr@temp         \def\@tempa{\TeXr@label\@arabic}\else
  \ifx \@sptoken\TeXr@temp \let\@tempa\TeXr@space         \else
  \ifx \bgroup\TeXr@temp   \let\@tempa\TeXr@group         \else
  \ifx \@@@\TeXr@temp      \let\@tempa\@gobble          \else
                           \let\@tempa\TeXr@other
```

Hook for possible extensions

```
                          \TeXr@hook
                  \fi\fi\fi\fi\fi\fi\fi\fi
  \@tempa}
```

`\TeXr@hook`

```
\providecommand\TeXr@hook{}
```