The cals package*

Oleg Parashchenko
olpa@uucode.com

November 23, 2016

1 Introduction

The `cals` package is a set of macros to typeset multipage tables with repeatable headers and footers, with cells spanned over rows and columns. Decorations are supported: padding, background color, width of separation rules. The code is compatible with multicols and bidi.

The work is released to public (LaTeX license) by `bitplant.de` GmbH, a company which provides technical documentation services to industry.

2 Usage

The users' guide is a separate document, published in TUGboat 2011:2: `http://tug.org/TUGboat/tb32-2/tb101parashchenko.pdf`. There are examples on CTAN: `https://www.ctan.org/pkg/cals`.

The most important feature: the table (its rows) must start in a vertical mode, the cells content should switch to a horizontal mode. For using the commands with the `@` symbol you might need to say `\makeatletter` first.

Please post questions and suggestions to the `texhax` mailing list (`http://tug.org/mailman/listinfo/texhax`) or on TeX-LaTeX Stack Exchange site (`http://tex.stackexchange.com/`) with the tag `cals`.

Summary of the user interface:

```
\begin{calstable} % [n|l|c|r]
\colwidths{{100pt}{200pt}}
\brow \cell{a} \cell{b} \erow
\end{calstable}
```

Table elements: `\thead`, `\tfoot`, `\tbreak{\penalty-10000}`, `\lastrule`.

Table alignment: `l`, `c`, `r` for left, center and right; `n` for none, using the default `\leftskip` and `\rightskip`.

---

[*] This document corresponds to cals CALS, dated 2016/11/23.

In-cell alignment: `\alignL`, `\alignC`, `\alignR`, `\vfill`.

Padding: lengths `\cals@paddingL` (`...T,R,B`), set by `\cals@setpadding{Ag}`, baseline alignment `\cals@paddingD`, set by `\cals@setcellprevdepth{Al}`.

Color: `\cals@bgcolor`.

Rules: `\cals@cs@width`, `\cals@framecs@width`, `\cals@rs@width`, `\cals@framers@width`, `\cals@bodyrs@width`. Overrides: `\cals@borderL` (`...T,R,B`).

Hooks: `\cals@AtBeginTable`, `\cals@AtEndTable`, `\cals@AtBeginCell`, `\cals@AtEndCell`.

Spanning: `\nullcell`, `\spancontent`.

3 Implementation

What happens. `\cell` creates a table cell, puts it to the current row and updates decorations. At the end of the row (`\erow`) we have the box `\cals@current@row`, the box `\cals@current@cs`—column separation and cells background—and the macros `\cals@current@rs@above` and `\cals@current@rs@below`—all the required data to typeset row separation. Before dispatching the row, all the cells are repacked to the common height. The row dispatcher (`\cals@row@dispatch`) usually just uses `\cals@issue@row`, which outputs `current@cs`, then joins the previous row `cs@below` with the current row `rs@above` and typesets the resulting row separation, and finally prints the row itself. If a table break is required, the dispatcher backups the current row and first typesets the table footer, a page break and the table header. In case of a row span, the set of the rows is converted to one big row.

I tried to code as good and robust as I can. In particular, the package contains unit tests. However, being an unexperienced TEX programmer, I could write bad code, especially in the section “List list of tokens”. Do not hesitate to send me suggestions and corrections, also in the use of English.

The description is split on two parts: main functionality and decorations. The first part is bottom-up: creating cells, collecting cells to a row, dispatching a row, top-level table elements. The second part starts with the common code, then explains in-row decorations (column separation and cells background) and between-row decorations (row separation).

3.1 Creating cells

`\cals@cell`
`\cals@cell@end`

Creates an individual cell before socialization into a table. Content of the cell is typeset inside a group. Execution continues in `\cals@celll@end`. Parameters:

1. Width of the cell
2. Vertical correction: when we have a rowspan, the cell is created while processing the last row. The vertical correction is required to raise the text back to the first row of the rowspan.
3. (Implicit parameter.) Content. It is important that it contains a switch to the horizontal mode, otherwise horizontal dimensions of the cell will be incorrect.

Using an implicit parameter instead of putting it to a macro parameter is probably a premature optimization.

```
\newcommand\cals@cell[3]{}
\def\cals@cell#1#2{%
```

Start immediately with `\vbox` to allow `\setbox0=\cals@cell{...}` construction. Later, white integrating the cell into a row, the content will be unvboxed and put to a vbox of the row height.

```
\vbox\bgroup%
```

Implicitely sets the width and the horizontal paddings of the cell. These settings come into effect on switch from the restricted vertical mode (our `\vbox`) to the horizontal mode. Therefore, the content must force such switch, otherwise the code fails.

```
\hsize=#1
\linewidth=#1
\leftskip=\cals@paddingL %
\rightskip=\cals@paddingR %
```

Vertical correction and top padding

```
\ifdim #2>0pt %
 \vskip-#2
\fi
\vskip\cals@paddingT %
```

Tuning the top padding. First, compensate the `\parskip`, which appears on the mode switch. Second, adjusts baselineskip, so in the case of the right preliminimary setup, the top of the letters "Al" touches the padding border. Meanwhile, setting prevdepth aligns the baselines of the first text lines of the row cells.

```
\vskip-\parskip %
\prevdepth=\cals@paddingD %
```

Finally, the content. And the switch to the horizontal mode (we hope).

We want more work afrer typesetting the content, but it is not desirable to collect all the tokens. Instead, start a group and use `\aftergroup` to finish typesetting. For more explanations, see "TEX by Topic", Chapter 12 "Expansion".

```
\bgroup\aftergroup\cals@cell@end
\cals@AtBeginCell\let\next=% eat '{' of the content
}%{Implicit content}

```

The infinite glue before the bottom padding is useful later, when we will reheight the cells in a row.

```
\def\cals@cell@end{\vfil\vskip\cals@paddingB
\cals@AtEndCell\egroup % finish vbox
```

Call the caller

```
\cals@celll@end}
```

`\cell` Creates a cell and appends it to the hbox `\cals@current@row`.

```
\newcommand\cell[1]{}
\def\cell{%
```

Get the width of the cell and typeset it to the box 0. The execution flow is: `\cell` to `\cals@cell` to `\cals@cell@end` to `\cals@celll@end`.

```
\llt@rot\cals@colwidths
\let\cals@cell@width=\llt@car
\setbox0=\cals@cell\cals@cell@width{0pt}%
}
```

`\cals@AtBeginCell` `\cals@AtEndCell` Additional code to be executed at the begin and at the end of a cell. An use case is a hook for pdfsync: `\def\cals@AtBeginCell{\pdfsyncstart}`. I am not sure if the end-hook is useful because all the changes are local for the cell group, but decided to retain it for symmetry.

```
\let\cals@AtBeginCell=\relax
\let\cals@AtEndCell=\relax
```

`\cals@width@cell@put@row` Implicit setting of the cell width can fail (example is an empty cell). In this case, force the width explicitely. Then put the cell to the current row. This code should be a part of `\cals@celll@end`, but due to `\spancontent` is in a separate macro.

```
\newcommand\cals@width@cell@put@row{%
\ifdim \cals@cell@width=\wd0 \relax \else \wd0=\cals@cell@width \fi
\setbox\cals@current@row=\hbox{\unhbox\cals@current@row\box0 }}%
```

`\cals@celll@end` After a cell is typeset to the box 0, execution continues here (see notes to `\cell`). Update the current row and its decorations.

```
\newcommand\cals@celll@end{%
\cals@width@cell@put@row
\cals@decor@next\cals@cell@width}
```

`\spancontent` Typesets a spanned cell (the content is in the implicit argument) and puts it to the current row. The width and height correction are already calculated, the decorations are also already added.

```
\newcommand\spancontent[1]{}
\def\spancontent{%
\let\cals@tmp=\cals@celll@end
\let\cals@cell@width=\cals@span@width
\def\cals@celll@end{%
  \cals@width@cell@put@row%
  \let\cals@celll@end=\cals@tmp}%
\setbox0=\cals@cell{\cals@span@width}{\cals@span@height}%
}%{Implicit content}
```

3.1.1 Cell padding

`\cals@setpadding` `\cals@paddingL` `\cals@paddingR` `\cals@paddingT` `\cals@paddingB` Calculates and sets the cell padding. It seems that a good value is the half of the font size, calculated as the full height of a box with the content #1. The calstable environment uses the letters "Ag".

```
\newskip\cals@paddingL
\newskip\cals@paddingR
\newskip\cals@paddingT
\newskip\cals@paddingB

\newcommand{\cals@setpadding}[1]{%
\setbox0=\hbox{#1}%
\dimen0=\ht0 \advance\dimen0 by \dp0 \divide\dimen0 by 2
\cals@paddingL=\dimen0 \relax
\cals@paddingR=\cals@paddingL
\cals@paddingT=\cals@paddingL
\cals@paddingB=\cals@paddingL
}
```

`\cals@setcellprevdepth`
`\cals@paddingD`

The function `\cals@cell` uses the length `\cals@paddingD` to tune the top padding. The macro `\cals@setcellprevdepth` calculates and sets this parameter, so that a box with the content #1 touches the padding border. The calstable environment uses the letters “Al”.

```
\newdimen\cals@paddingD

\newcommand{\cals@setcellprevdepth}[1]{%
\setbox0=\vbox{\prevdepth=0pt #1}%
\setbox1=\vbox{#1}%
\dimen0=\ht0 \advance\dimen0 by \dp0 %
\advance\dimen0 by -\ht1 \advance\dimen0 by -\dp1%
\cals@paddingD=\dimen0 }
```

`\alignL`
`\alignC`
`\alignR`

To align the table cell text left, center or right we add or remove vfill-part of the left and right padding. Executed by assigning skip to dimen.

```
\newcommand\alignL{%
\cals@vfillDrop\cals@paddingL
\cals@vfillDrop\cals@paddingR}

\newcommand\alignC{%
\cals@vfillAdd\cals@paddingL
\cals@vfillAdd\cals@paddingR}

\newcommand\alignR{%
\cals@vfillAdd\cals@paddingL
\cals@vfillDrop\cals@paddingR}
```

`\cals@vfillAdd`
`\cals@vfillDrop`

Add or remove the `vfill`-part of a skip. Retain the existing value if possible.

```
\newcommand\cals@vfillAdd[1]{\ifnum\gluestretchorder#1>1\relax\else
\dimen0=#1\relax #1=\dimen0 plus 1fill\relax \fi}
\newcommand\cals@vfillDrop[1]{\ifnum\gluestretchorder#1>0\relax
\dimen0=#1\relax #1=\dimen0\relax \fi}
```

3.2 From cells to a row

`\cals@current@row`

Rows are first typeset to this hbox.

```
\newbox\cals@current@row
```

`\colwidths`
`\cals@colwidths`

The macro `\cals@colwidths` contains a list of column widths. The user sets it through the API macro `\colwidths`, which performs expansion. The list is alive, it is rotated after a cell is finished, so the width of the next cell is always the first element.

```
\newcommand\colwidths[1]{}
\def\colwidths#{\edef\cals@colwidths}
\def\cals@colwidths{{100pt}}
```

Initially, I planned to use `\row{...}` to create a row. In order to perform actions at the end of the row I used the aftergroup-trick, like for `\cell` command. Unfortunately, the decorations were lost after the group finished. I tried to save them to global temporary macros at the end of each cell, but the saving list was big. Finally, I decided that the construction `\brow...\erow` is much easier to implement.

`\brow`

Starts a row. Resets the rowspan markers, `\cals@current@row` and decorations.

```
\newcommand\brow{%
\cals@updateRspanMarkers
\setbox\cals@current@row=\hbox{}%
\cals@decor@begin}
```

`\erow`

Finishes a row. All the cells are re-layouted to the row height. Decorations are finalized, and the row is dispatched.

```
\newcommand\erow{%
\cals@reheight@cells\cals@current@row
\cals@last@row@height=\ht\cals@current@row\relax
\cals@decor@end\cals@lastWidth
\ht\cals@current@cs=\ht\cals@current@row
\cals@row@dispatch
}
```

`\cals@reheight@cells`

Re-heights all the boxes of a row. Retains the widths of these boxes.

```
\newcommand\cals@reheight@cells[1]{%
\dimen0=\ht#1\relax
\setbox2=\hbox{}%
\def\next{%
 \setbox4=\lastbox
 \ifvoid4
   \def\next{\global\setbox2=\box2}%
 \else
   \dimen4=\wd4
   \setbox4=\vbox to \dimen0{\unvbox4}%
   \ifdim \dimen4=\wd4 \relax \else \wd4=\dimen4 \fi
   \setbox2=\hbox{\box4\unhbox2 }%
\fi
\next}%
\setbox0=\hbox{\unhbox#1\next}%
\setbox#1=\box2 }
```

3.3 Spanned cells

The technical approach:

- Spanning is started in the left top corner using the command `\nullcell{lt...}`
- The spanned cell is split on the table cells using the command `\nullcell`. These nullcells are responsible for correct decorations and for calculating the big cell dimensions.
- The content of the spanned cells comes in the end, in the right bottom corner.

It is possible to have several active spans at once, therefore we have to remember them. I use a queue. Each time a left column of a span is started, we take span data from the queue. After the right column of the span, we put the data back to the queue, to the end. Probably it is not obvious at the first look (at least, I needed time to find this simple idea) that this rotating queue is always right: when spanning (its left column) starts again, the beginning of the queue always contains data for exactly this spanning.

`\cals@spanq@heights` The queue for height tracking. In the first version I also had a queue for decorations, but later cancelled it. I use `\def` instead of `\newcommand` in order that `\show` prints "`macro`" instead of "`\long macro`". The latter breaks unit tests.

```
\def\cals@spanq@heights{}
```

`\cals@span@get` Gets `\cals@span@height` from the queue.

```
\newcommand\cals@span@get{%
\llt@decons\cals@spanq@heights \cals@span@height=\llt@car\relax}
```

`\cals@span@put` Puts `\cals@span@height` to the queue.

```
\newcommand\cals@span@put{%
\edef\cals@tmp{\the\cals@span@height}\llt@snoc\cals@spanq@heights\cals@tmp}
```

`\nullcell` The big spanned cell is split on the table cells, which are identified by `\nullcell`s. The task is to produce correct decorations and to track the parameters of the spanned cell.

- Decorations: if defined, the background color is always added to the column separation row.
- Decorations: the borders are set to 0pt (disabled), except for the borders which are requested by the parameter of `\nullcell`: `l` for the left border, `t` for the top, `r` for the right, `b` for the bottom.

- More precisely, the letters in the argument are not to specify which decorations to use, but to specify the location of the small cell in the big cell. The use for decorations is just an useful side-effect.
- Action `l`: take the span data from the queue.
- Action `r`: update the height of the current span, put the data to the queue.
- Action `b`: do not put an empty box to the current row. Instead, accumulate the width of the current span. (Preparation for `\spancontent`.)

```
\newcommand\nullcell[1]{%
```

First of all, parse the argument and set the if-commands `\cals@span@ifX`. Then get the width of the cell.

```
\let\cals@span@ifL=\cals@iffalse
\let\cals@span@ifT=\cals@iffalse
\let\cals@span@ifR=\cals@iffalse
\let\cals@span@ifB=\cals@iffalse
\def\next##1{\ifx\relax##1\let\next=\relax \else
 \expandafter\let\csname cals@span@if##1\endcsname=\cals@iftrue \fi
 \next}%
\uppercase{\next #1}\relax
\llt@rot\cals@colwidths \let\cals@nullcell@width=\llt@car
```

Action "l": update the height, supress the borders, set the rowspan markers.

```
\cals@span@ifL\iftrue
 \cals@span@ifT\iftrue
   \cals@span@height=0pt %
 \else
   \cals@span@get
   \advance\cals@span@height by \cals@last@row@height\relax
 \fi
 \cals@span@ifB\iftrue \else
  \let\cals@ifInRspan=\cals@iftrue
  \let\cals@ifLastRspanRow=\cals@iffalse
 \fi
 \let\cals@span@borderL=\cals@borderL \let\cals@span@borderT=\cals@borderT
 \let\cals@span@borderR=\cals@borderR \let\cals@span@borderB=\cals@borderB
\fi
```

Action "r": put the data to the queue, unless in the end of the spanning.

```
\cals@span@ifR\iftrue
 \cals@span@ifB\iftrue \relax \else \cals@span@put \fi
\fi
```

Update the current row or calculate the span width (in the case of the bottom row).

```
\cals@span@ifB\iftrue
 \cals@span@ifL\iftrue
  \cals@span@width=\cals@nullcell@width\relax
 \else
```

```
  \advance\cals@span@width by \cals@nullcell@width\relax
 \fi
\else
 \setbox\cals@current@row=\hbox{%
   \unhbox\cals@current@row
   \vbox{\hbox to\cals@nullcell@width{}\vfil}}%
\fi
```

Update decorations

```
\cals@span@ifL\iftrue \let\cals@borderL=\cals@span@borderL
  \else \def\cals@borderL{0pt}\fi
\cals@span@ifT\iftrue \let\cals@borderT=\cals@span@borderT
  \else \def\cals@borderT{0pt}\fi
\cals@span@ifR\iftrue \let\cals@borderR=\cals@span@borderR
  \else \def\cals@borderR{0pt}\fi
\cals@span@ifB\iftrue \let\cals@borderB=\cals@span@borderB
  \else \def\cals@borderB{0pt}\fi
\cals@decor@next\cals@nullcell@width
\let\cals@borderL=\cals@span@borderL \let\cals@borderR=\cals@span@borderR
\let\cals@borderT=\cals@span@borderT \let\cals@borderB=\cals@span@borderB
}
```

`\cals@span@width` `\cals@span@height` The width of the span cell. The height of the spanned cell (without the last row).

```
\newdimen\cals@span@width
\newdimen\cals@span@height
```

`\cals@ifInRspan` Set to `\cals@iftrue` if the current row is a part of a row span. Otherwise `\cals@iffalse`.

`\cals@ifLastRspanRow` Set to `\cals@iftrue` if the current row is the last row of of a row span. Otherwise `\cals@iffalse`.

`\cals@updateRspanMarkers` Resets the span markers, which are later updated by `\nullcell` to the correct state for the current row.

```
\newcommand\cals@updateRspanMarkers{%
\ifx \empty\cals@spanq@heights
 \let\cals@ifInRspan=\cals@iffalse
\else
 \let\cals@ifInRspan=\cals@iftrue
\fi
\let\cals@ifLastRspanRow=\cals@iftrue}
```

3.4 Row dispatcher

`\cals@row@dispatch` Depending if the current row has a rowspan cell or not, the execution is different.

```
\newcommand\cals@row@dispatch{%
\ifx b\cals@current@context
 \cals@ifInRspan\iftrue
  \cals@row@dispatch@span
 \else
  \cals@row@dispatch@nospan
 \fi
\else
 \cals@row@dispatch@nospan
\fi}
```

`\cals@row@dispatch@nospan` After a row is typeset in a box, this macro decides what to do next. Usually, it should just add decorations and output the row. But if a table break is required, it should put the current row to backup, typeset the footer, the break, the header and only then the row from the backup. Summary of main parameters:

- rowsep from the last row (`\cals@last@rs`) and the last context (`\cals@last@context`)
- current row (`\cals@current@row`), its decorations (`\cals@current@cs`, `\cals@current@rs@above`, `\cals@current@rs@below`) and context (`\cals@current@context`)

```
\newcommand\cals@row@dispatch@nospan{%
```

The header and footer rows are always typeset without further considerations.

```
\let\cals@last@context@bak=\cals@last@context
\ifx h\cals@current@context \else
\ifx f\cals@current@context \else
```

In the body, if a break is required: do it.

```
\cals@ifbreak\iftrue
 \setbox\cals@backup@row=\box\cals@current@row
 \setbox\cals@backup@cs=\box\cals@current@cs
 \let\cals@backup@rs@above=\cals@current@rs@above
 \let\cals@backup@rs@below=\cals@current@rs@below
 \let\cals@backup@context=\cals@current@context
 \cals@tfoot@tokens
 \lastrule
 \cals@issue@break
 \cals@thead@tokens
 \setbox\cals@current@row=\box\cals@backup@row
 \setbox\cals@current@cs=\box\cals@backup@cs
 \let\cals@current@rs@above=\cals@backup@rs@above
 \let\cals@current@rs@below=\cals@backup@rs@below
 \let\cals@current@context=\cals@backup@context
\fi\fi\fi
```

Typeset the row. If the width of the row is more than hsize, then the issue-code should not fit the row to hsize.

```
\ifdim\wd\cals@current@row>\hsize\relax
\def\cals@tohsize{}%
\fi
\cals@issue@row
```

Consider a table such that thead+row1 do not fit to a page (see the unit test `regression/test_010_wrongbreak`). Without the next code, the following happens: thead and row1 are typeset, but the output procedure is not executed yet.

Therefore, when row2 is ready, we detect that a table break is required and create it. Then the output procedure moves thead+row1 on the next page. The result: thead and row1 on one page, row2 and the rest on the next page instead of the whole table on one page. Solution: force a run of the output procedure after the first row of a table chunk.

```
\ifx b\cals@last@context
 {\dimen0=\pagetotal\relax
 \advance\dimen0 by \cals@tfoot@height\relax
 \advance\dimen0 by -\pagegoal
 \ifdim\dimen0>0pt\relax
   \vskip\dimen0
   \penalty9999 % with 10000, the output page builder is not called
   \vskip-\dimen0\relax
 \fi
 }%
\fi
}
```

`\cals@row@dispatch@span`

The only specific thing to rowspanned rows is that we should not allow breaks between the rows in one group. We put these rows to one box, and process this big box as a big row.

```
\newcommand\cals@row@dispatch@span{%
```

Output the row to the backup box. If the row is the first row in the span, let its decorations will be the decorations for the future big row. Also, reset the values of leftskip and rightskip to avoid adding them twice, once in a individual row, and once to the common span box.

```
\ifvoid\cals@backup@row
 \setbox\cals@backup@row=\vbox{\box\cals@current@row}%
 \setbox\cals@backup@cs=\box\cals@current@cs
 \let\cals@backup@rs@above=\cals@current@rs@above
 \let\cals@backup@last@rs@below=\cals@last@rs@below
 \let\cals@backup@context=\cals@last@context
 \cals@backup@leftskip=\leftskip\relax
 \cals@backup@rightskip=\rightskip\relax
 \let\cals@backup@tohsize=\cals@tohsize
 \leftskip=0pt\relax \rightskip=0pt\relax \def\cals@tohsize{}%
\else
 \setbox\cals@backup@row=\vbox{\unvbox\cals@backup@row
  \cals@issue@row}%
\fi
\let\cals@last@rs@below=\cals@current@rs@below
\let\cals@last@context=\cals@current@context
```

If this is the last row of the span, create the fake big row and use the normal dispatcher.

```
\cals@ifLastRspanRow\iftrue
 \setbox\cals@current@row=\box\cals@backup@row
 \setbox\cals@current@cs=\box\cals@backup@cs
 \let\cals@current@rs@above=\cals@backup@rs@above
 \let\cals@last@rs@below=\cals@backup@last@rs@below
 \let\cals@last@context=\cals@backup@context
 \leftskip=\cals@backup@leftskip
 \rightskip=\cals@backup@rightskip
 \let\cals@tohsize=\cals@backup@tohsize
 \cals@row@dispatch@nospan
\fi
}
```

`\cals@backup@row` `\cals@backup@cs` Boxes and skips for backup.

```
\newbox\cals@backup@row
\newbox\cals@backup@cs
\newskip\cals@backup@leftskip
\newskip\cals@backup@rightskip
```

To decide on table breaks and row separation decorations, we need to trace context.

`\cals@current@context` The context of the current row. Possible values, set as a ”`\let`” to a character:

- n: no context, should not happen when the value is required
- h: table header
- f: table footer
- b: table body

`\cals@last@context` The context of the previous row. Possible values, set as a ”`\let`” to a character:

- n: there is no previous row (not only the start of a table, but also the start of a table chunk)
- h, f, b: table header, footer, body
- r: a last rule of the table (or its chunk) is just output. This status is used to allow multiple calls to `\lastrule`. Probably the use of `current` instead of `last` is more logical, but using `last` is more safe. Who knows if an user decides to use `\lastrule` somewhere in a middle of a table.

`\cals@ifbreak` Table breaks can be manual or automatic. The first is easy, the second is near to impossible if we take into account table headers and footer. The following heuristic seems good.

Check if the current row plus the footer fits to the rest of the page. If not, a break is required. This approach is based on two assumptions:

- the height of the footer is always the same, and
- any body row is larger than the footer.

More precise and technical description: `\cals@ifbreak` decides if an automatic table break is required and leaves the macro `\cals@iftrue` (yes) or `\cals@iffalse` (no) in the input stream. If the user sets `\cals@tbreak@tokens` (using `\tbreak`), break is forced. Otherwise, no break is allowed:

- inside a box (= inside a float)
- In the header
- In the footer
- Immediately after the header
- At the beginning of a chunk of a table.

Otherwise break is recommended when the sum of the height of the current row and of the footer part is greater as the rest height of the page. The implicit first parameter is used for if-fi balancing, see `\cals@iftrue`.

```
\newcommand\cals@ifbreak[1]{}
\def\cals@ifbreak{%
\let\cals@tmp=\cals@iffalse
\let\cals@tmpII=\cals@iftrue
\ifx\relax\cals@tbreak@tokens
 \ifinner\else
  \ifx h\cals@current@context \else
   \ifx f\cals@current@context \else
    \ifx h\cals@last@context \else
     \ifx n\cals@last@context \else
      \dimen0=\pagetotal\relax
      \advance\dimen0 by \ht\cals@current@row\relax
      %\showthe\ht\cals@current@row\relax
      \ifx \cals@tfoot@tokens\relax \else
        %\show\cals@tfoot@height\relax
        \advance\dimen0 by \cals@tfoot@height\relax
      \fi
      %\showthe\dimen0\relax
      \ifdim \dimen0>\pagegoal\relax
        \let\cals@tmp=\cals@tmpII
      \fi
 \fi\fi\fi\fi\fi
\else \let\cals@tmp=\cals@tmpII % tbreak@tokens
\fi
\cals@tmp}
```

`\cals@issue@break` By default, force a page break, otherwise use user's tokens set by `\tbreak`.

```
\newcommand\cals@issue@break{\ifx \relax\cals@tbreak@tokens \penalty-10000 %
\else \cals@tbreak@tokens \fi
\let\cals@tbreak@tokens=\relax
\let\cals@last@context=n}
```

`\cals@set@tohsize`
`\cals@tohsize`

Table row contains not only the row itself, but also `\leftskip` and `\rightskip`. Now the dilemma. If the row is just `\hbox`, than the glue component is ignored, and the table always aligned left. On the other side, if the row is `\hbox to \hsize`, then the user gets underfulled boxes. A simple solution is to switch on and off the `hsize`-part depending on the skips.

```
\newcommand\cals@tohsize{}
\newcommand\cals@set@tohsize{\def\cals@tohsize{}%
\ifnum\gluestretchorder\leftskip>0\relax \def\cals@tohsize{to \hsize}\fi
\ifnum\gluestretchorder\rightskip>0\relax \def\cals@tohsize{to \hsize}\fi
}
```

`\cals@activate@rtl`
`\cals@deactivate@rtl`
`\cals@hbox`

For bidi support, use `\hboxR` instead of `\hbox`. Actually, more is required for bidi support, and these macros are retained only as "legacy". Otherwise I'd move the code into the beginning of ‘calstable‘.

```
\newcommand\cals@hbox{}
\newcommand\cals@activate@rtl{\let\cals@hbox=\hboxR}
\newcommand\cals@deactivate@rtl{\let\cals@hbox=\hbox}
\cals@deactivate@rtl
```

`\cals@issue@rowsep@alone`

Typesets the top (or bottom) frame of a table: combines `\cals@current@rs@above` and `\cals@framers@width` and outputs the row separator.

```
\newcommand\cals@issue@rowsep@alone{%
\setbox0=\cals@hbox\cals@tohsize{%
 \cals@hskip@lr\leftskip\rightskip
 \cals@rs@sofar@reset
 \cals@rs@joinOne\cals@framers@width\cals@current@rs@above
 \cals@rs@sofar@end
 \cals@hskip@lr\rightskip\leftskip}%
\ht0=0pt \dp0=0pt \box0 }
```

`\cals@issue@rowsep`

Combine row separations `\cals@last@rs@below` and `\cals@current@rs@above`, taking into considiration the width of the rule:

- n to h, f, b (the top frame): use `\cals@framers@width` and ignore `last@rs@below` because we don't have it
- h to h, b to b, f to f (the usual separator): use `\cals@rs@width`
- for all other combinations (header to body, body to footer), including impossible: use `\cals@bodyrs@width`

```
\newcommand\cals@issue@rowsep{%
\ifx n\cals@last@context \cals@issue@rowsep@alone \else
 \ifx \cals@last@context\cals@current@context
   \let\cals@tmpIII=\cals@rs@width     \else
   \let\cals@tmpIII=\cals@bodyrs@width \fi
 \setbox0=\cals@hbox\cals@tohsize{%
  \cals@hskip@lr\leftskip\rightskip
  \cals@rs@sofar@reset
  \cals@rs@joinTwo\cals@tmpIII\cals@last@rs@below\cals@current@rs@above
  \cals@rs@sofar@end
  \cals@hskip@lr\rightskip\leftskip}%
 \ht0=0pt \dp0=0pt \box0 %
\fi}
```

`\cals@last@row@height` For spanning support, we need to remember the height of the last row

```
\newdimen\cals@last@row@height
```

`\cals@issue@row` Typesets the current row and its decorations, then updates the last context. Regards `\leftskip` and `\rightskip` by putting them inside the row.

```
\newcommand\cals@issue@row{%
```

Decorations: first the column separation, then the row separation.

```
\nointerlineskip
\setbox0=\vtop{\cals@hbox\cals@tohsize{\cals@hskip@lr\leftskip\rightskip
\box\cals@current@cs \cals@hskip@lr\rightskip\leftskip}}%
\ht0=0pt\relax\box0
\nointerlineskip
\cals@issue@rowsep
\nointerlineskip
```

Output the row, update the last context.

```
\cals@hbox\cals@tohsize{\cals@hskip@lr\leftskip\rightskip
\box\cals@current@row \cals@hskip@lr\rightskip\leftskip}%
\let\cals@last@rs@below=\cals@current@rs@below
\let\cals@last@context=\cals@current@context
\nobreak}
```

3.5 Table elements

`\calstable` Setup the parameters and let the row dispatcher to do all the work.

```
\newenvironment{calstable}[1][\cals@table@alignment]{%
\if@RTL\@RTLtabtrue\cals@activate@rtl\fi
\let\cals@thead@tokens=\relax
\let\cals@tfoot@tokens=\relax
\let\cals@tbreak@tokens=\relax
\cals@tfoot@height=0pt \relax
\let\cals@last@context=n%
\let\cals@current@context=b%
\parindent=0pt \relax%
\cals@setup@alignment{#1}%
\cals@setpadding{Ag}\cals@setcellprevdepth{Al}\cals@set@tohsize%
%% Alignment inside is independent on center/flushright outside
\parfillskip=0pt plus1fil\relax
\let\cals@borderL=\relax
\let\cals@borderR=\relax
\let\cals@borderT=\relax
\let\cals@borderB=\relax
```

Bug fix for `http://tex.stackexchange.com/questions/167400/fancyhdr-and-cals-vertical-merge` Table inside a table is ok, but when there are 1) page break inside a table in the text flow, and 2) a table with a vertically straddled cell is created in the output procedure, then this table inside table needs additional cleaning.

```
\setbox\cals@backup@row=\box\voidb@x\relax
\cals@AtBeginTable
}{% End of the table
\cals@tfoot@tokens\lastrule\cals@AtEndTable}
```

`\cals@AtBeginTable`
`\cals@AtEndTable`

Callbacks for more initialization possibilities.

```
\newcommand\cals@AtBeginTable{}%
\newcommand\cals@AtEndTable{}%
```

`\lastrule`

Typesets the last rule (bottom frame) of a table chunk. Repeatable calls are ignored. Useful in the macro `\tfoot`.

```
\newcommand\lastrule{%
\ifx r\cals@last@context \relax \else
 \let\cals@last@context=r%
 \nointerlineskip
 \let\cals@current@rs@above=\cals@last@rs@below\cals@issue@rowsep@alone%
\fi}
```

`\thead`

Table: the header. Remember for later use, typeset right now.

```
\newcommand\thead[1]{%
\def\cals@thead@tokens{\let\cals@current@context=h%
#1\let\cals@current@context=b}%
\cals@thead@tokens}
```

`\tfoot`

Table: the footer. Remember for later use. Right now, typeset to a box to calculate an expected height for the table breaker `\cals@ifbreak`.

```
\newcommand\tfoot[1]{%
\def\cals@tfoot@tokens{\let\cals@current@context=f#1}%
\setbox0=\vbox{\cals@tfoot@tokens}%
\cals@tfoot@height=\ht0 \relax}
```

`\cals@tfoot@height`

The height of the footer.

```
\newdimen\cals@tfoot@height
```

`\tbreak`

Table: force a table break. Argument should contain something like `\penalty-10000` .

```
\newcommand\tbreak[1]{\def\cals@tbreak@tokens{#1}}
```

`\cals@table@alignment`

The default alignment of tables in the text flow. Doesn't affect the text alignment inside cells.

- `n`: no settings, the default `\leftskip` and `\rightskip` are used
- `l`: align left
- `c`: align center

- `r`: align right

This setting is appeared in the version 2.3. Earlier versions worked as it were `n`.

```
\newcommand\cals@table@alignment{l}
```

3.6 List list of tokens

Two-dimensional arrays of tokens, or lists of lists of tokens.

Format of the list:

```
{...tokens1...}{...tokens2...}...{...tokensN...}
```

Token manipulation should not belong to the “cals” package, and the macros from this section have the prefix `llt@` instead of `cals@`. Probably it is better to use some CTAN package, but initially the llt-code was small and simple, so I did not want dependencies, and now I do not want to replace working code with something new.

In comments to these functions, a parameter of type *token list* is a macro which will be expanded once to get the tokens, and *list list* is a macro which stores the two-dimensional array.

An example of use:

```
\def\aaa{aaa}
\def\bbb{bbb}
\def\ccc{ccc}
\def\lst{}         % empty list
\llt@cons\bbb\lst  % \lst -> "{bbb}"
\llt@snoc\lst\ccc  % \lst -> "{bbb}{ccc}"
\llt@cons\aaa\lst  % \lst -> "{aaa}{bbb}{ccc}"
\llt@decons\lst    % \llt@car -> "aaa", \lst -> "{bbb}{ccc}"
\llt@rot\lst       % \llt@car -> "bbb", \lst -> "{ccc}{bbb}"
```

`\llt@cons` Prepends the token list #1 to the list list #2. Corrupts the token registers 0 and 2.

```
\def\llt@cons#1#2{%
\toks0=\expandafter{#1}%
\toks2=\expandafter{#2}%
\edef#2{\noexpand{\the\toks0}\the\toks2 }%
}
```

`\llt@snoc` Appends the token list #2 to the list list #1 (note the order of parameters). Macro corrupts the token registers 0 and 2.

```
\def\llt@snoc#1#2{%
\toks0=\expandafter{#1}%
\toks2=\expandafter{#2}%
\edef#1{\the\toks0 \noexpand{\the\toks2}}%
}
```

`\llt@car` A token list, set as a side-effect of the list deconstruction and rotation functions.

`\llt@decons` List deconstruction. The first item is removed from the list list #1 and its tokens are put to the token list `\llt@car`. Corrupts the token register 0. Undefined behaviour if the list list has no items.

The actual work happens on the `\expandafter` line. It's hard to explain, let me show the macro expansion, I hope it's self-explaining.

```
\expandafter\llt@decons@open\lst}      -->
\llt@decons@open{aaa}{bbb}{ccc}{ddd}}  -->
\def\llt@car{aaa} \toks0=\llt@opengroup {bbb}{ccc}{ddd}}  -->
\def\llt@car{aaa} \toks0={{bbb}{ccc}{ddd}}
```

Why I use `\let\llt@opengroup={` inside the definition? Only to balance the number of opening and closing brackets. Otherwise TeX will not compile the definition.

Initially I tried to use the following helper:

```
\def\decons@helper#1#2\relax{%
  \def\llt@car{#1}%
  \def\list{#2}}
```

If a call is `\decons@helper{aaa}{bbb}{ccc}\relax` then all is ok, the helper gets: #1 is `aaa` and #2 is `{bbb}{ccc}`.

Unfortunately, if the list has two items and the call is `\decons@helper{aaa}{bbb}}\relax`, then the helper gets: #1 is `aaa` and #2 is `bbb` instead of `{bbb}`. The grouping tokens are lost, and we can't detect it.

```
\def\llt@decons@open#1{%
\def\llt@car{#1}%
\toks0=\llt@opengroup
}

\def\llt@decons#1{%
\let\llt@opengroup={%
\expandafter\llt@decons@open#1}%
\edef#1{\the\toks0}%
}
```

`\llt@rot` Rotates the list list #1. The first item becomes the last. Also, its tokens are saved to token list `\llt@car`. The second item becomes the first item, the third the second etc. Corrupts the token registers 0 and 2.

```
\def\llt@rot#1{%
\ifx#1\empty
\let\llt@car=\relax
\else
\llt@decons#1%
\llt@snoc#1\llt@car%
\fi
}
```

`\llt@desnoc` Very unefficient list deconstruction. The last item is removed from the list list #1 and its tokens are put to the token list `\llt@car`. Corrupts the token registers 0 and 2. Undefined behaviour if the list list has no items.

`\llt@desnocII` is used to hide `\if` from the loop.

```
\def\llt@desnocII#1{
\ifx\empty#1%
\let\llt@tmp=n%
\else
\llt@snoc{\llt@newlist}{\llt@car}%
\let\llt@tmp=y%
\fi
}

\def\llt@desnoc#1{%
\def\llt@newlist{}%
\loop
\llt@decons{#1}%
\llt@desnocII{#1}%
\if y\llt@tmp \repeat
\let#1=\llt@newlist}
```

4 Decorations

How much decoration requires a table? Initially I thought to implement a generic approach, so an user could extend the set of what is possible—for example, a dashed border instead of a solid one. But this is a hard task for me, therefore I switched back to the fixed set of properties. Indeed, the use case for cals tables is technical documentation, not a wanna-be designer showcases.

The fixed set of decoration properties:

- padding
- border thickness
- cancelled after some thinking: border color
- cell background

In the first version of this package, decorations could be defined for all cells in a row, for all cells in a column and finally specially for a cell. Unfortunately, this approach does not work for borders of multipage tables, when the decorations on a break are different to the internal decorations. Trying different workarounds, I finally found a descriptive and direct approach: define default decorations for a cell plus define decorations for the table frame.

How to decorate the common border of two cells? The following seems reasonable:

- The priority of settings: from the user, from the table frame, default. If cells have different priorities, use the highest one.
- When priorities are the same, use the maximal thinkness.

Imagine that the user uses only default decorations. Then all the internal vertical borders are the same, and all internal horizontal border are also the same. It took me time to understand this obvious thing, that for the default setup we need to define settings only for vertical and horizontal borders, not for all four borders.

Column separation and row separation are two very different creatures. The former is fixed after a row is processed, the latter could change somewhen later due to a table break.

4.1 Setters and getters

No more setter and getters provided to discourage cell formatting. At the moment, if you really need it, use the knowledge of the internal variables.

`\cals@cs@width` Width of column separators (vertical borders). For all the widths, `0pt` disables the rule.

`\cals@framecs@width` Width of the left and right table frame border.

`\cals@rs@width` Width of row separators (horizontal borders).

`\cals@framers@width` Width of the top and bottom table frame border.

`\cals@bodyrs@width` Width of row separators between the body and the header or the footer.

`\cals@bgcolor` Background color of the cells. If the macro is empty, it means no background.

```
\newcommand\cals@cs@width{.4pt}
\newcommand\cals@framecs@width{0pt}
\newcommand\cals@rs@width{.4pt}
\newcommand\cals@framers@width{0pt}
\newcommand\cals@bodyrs@width{1.2pt}
\newcommand\cals@bgcolor{}
```

`\cals@borderL`
`\cals@borderT`
`\cals@borderR`
`\cals@borderB`

Overrides for the widths of the cell borders (left, top, right or bottom). Macros, set to `\relax` by the `calstable` environment.

Padding parameters (`\cals@paddingL` etc) and related macros (`\alignC` etc) are defined near the macro `\cell`.

4.2 Decorations for a row

The whole code in this section is devoted to provide functionality for the functions `\cals@decor@begin`, `...@next`, `...@end`. After a row is ended, we have the following decorations:

- column separation: box `\cals@current@cs`

- rowsep specification for the top: macro `\cals@rs@above`
- rowsep specification for the bottom: macro `\cals@rs@below`

The high-level approach is obvious: we construct the decorations cell-by-cell. First, we calculate column separation, getting the width of the left and the right border. Then we use these values to update the border above and below. Unfortunately, there is a lot of small details. For example, as explained later, we construct the decorations with a delay, therefore the end-function is just a special sort of the next-function.

`\cals@decor@begin` Initialization.

```
\newcommand\cals@decor@begin{\cals@csrow@begin\cals@rs@spec@begin}
```

`\cals@decor@next` Updates the decorations. The argument is the width of the cell.

```
\newcommand\cals@decor@next[1]{%
\cals@csrow@nextcell{#1}\cals@borderL\cals@borderR\cals@bgcolor
\cals@rs@spec@next{#1}\cals@lastLeftWidth\cals@borderT\cals@borderB}
```

`\cals@decor@end` Finishes the decorations. Uses `\cals@lastLeftWidth`, which is the width of the last right border.

```
\newcommand\cals@decor@end{%
\cals@csrow@end
\cals@rs@spec@end\cals@lastLeftWidth}
```

4.3 Deciding on the width and color

`\cals@widthII`
`\cals@withWidthII` Calculate a final width from the default one (argument 1) and user-specified (argument 2, `\relax` means use the default), put it to the macro `\cals@width`. Also, the macro `withWidthII` start a conditional construction, true branch is executed when the width is not zero. Unused #3 is for if-fi balancing, see `\cals@iftrue`.

```
\newcommand\cals@widthII[2]{%
\ifx \relax#2\edef\cals@width{#1}%
        \else \edef\cals@width{#2}\fi}

\newcommand\cals@withWidthII[3]{%
\cals@widthII{#1}{#2}%
\ifdim \cals@width>0pt }
```

`\cals@withColorII` Calculate a final color from the default one (argument 1) and user-specified (argument 2, `\relax` means use the default), set the macro `\cals@color` to it. The both arguments must be macro names, not token lists. Start a conditional construction, true branch is executed when the color name is given (not empty). Unused #3 is for if-fi balancing, see `\cals@iftrue`.

```
\newcommand\cals@withColorII[3]{%
\ifx \relax#2\edef\cals@color{#1}%
        \else \edef\cals@color{#2}\fi
```

Reversing a condition. Based on `\ifnot` macro by David Kastrup posted to comp.text.tex 2 March 1998, Message-ID: `<m2en0lebuc.fsf@mailhost.neuroinformatik.ruhr-uni-boc`

```
\ifx \cals@color\empty \else\expandafter\expandafter\fi\iffalse\iftrue\fi}
```

`\cals@halfWidthToDimen` Puts the half of the width in `#2` to the dimension register `#1`.

```
\newcommand\cals@halfWidthToDimen[2]{%
\dimen#1=#2\relax \divide\dimen#1 by 2 }
```

`\cals@maxWidth` Of two widths, given as macros, selects the maximal and put the result to `\cals@width`. Takes care for `\relax`.

```
\newcommand\cals@maxWidth[2]{%
\ifx \relax#1\relax
  \ifx \relax#2\let\cals@width=\relax
          \else \edef\cals@width{#2}\fi
\else
  \ifx \relax#2\relax
    \edef\cals@width{#1}%
  \else
    \ifdim #1>#2 \edef\cals@width{#1}%
           \else \edef\cals@width{#2}\fi\fi\fi}
```

`\cals@iftrue`
`\cals@iffalse`
Balancing if-fi. The following does not work:

```
\let\next=\iftrue ...
\if ... \next ... \fi ... \fi
```

But this code does work:

```
\let\next=\cals@iftrue ...
\if ... \next\iftrue ... \fi ... \fi
```

We use `\iftrue` (or any other if-start), which is taken into account when scanning for the fi-else-if balance, but ignored when executed.

```
\def\cals@iftrue#1{\iftrue}
\def\cals@iffalse#1{\iffalse}
```

4.4 Column separation (colsep) and cell background

In-row decorations are the vertical borders between the cells and also the background color of the cells.

`\cals@cs@outOne` Typesets the background and the left border of a cell. Decorations have zero depth and undefined height. Parameters:

1. Width of the cell. The use of `\relax` avoids typesetting the cell itself, which is used when creating the right frame of a table.
2. Width of the border. 0pt is no border.

3. Color of the background. Empty macro is no color.

If some arguments are undefined (through `\relax`), global variables are used: `\cals@bgcolor` and `\cals@cs@width`.

Corrupts `dimen0`.

```
\newcommand\cals@cs@outOne[3]{%
```

Create the full-width background

```
\ifx \relax#1%
\else
 \cals@withColorII\cals@bgcolor{#3}\iftrue
  \textcolor{\cals@color}{\vrule depth0pt width#1 }%
  \hskip -#1\relax
 \fi
\fi
```

The border. I feel that overprinting the background is good, but I don't know why I think so.

```
\cals@withWidthII\cals@cs@width{#2}\iftrue
 \cals@halfWidthToDimen0 \cals@width %
 \hskip -\dimen0 %
 \vrule depth0pt width\cals@width\relax
 \hskip -\dimen0 %
\fi
```

We will need the actual width of the left border in a grand-grand-...-caller, when constructing a rowsep specification.

```
\let\cals@lastLeftWidth=\cals@width
```

Add width to skip to the next cell.

```
\ifx \relax#1\else \hskip#1 \fi
}
```

`\cals@current@cs` The box to store column separation.

```
\newbox\cals@current@cs
```

`\cals@csrow@begin`
`\cals@csrow@nextcell`
`\cals@csrow@end`

Constructs an hbox with colsep decorations. Call to the begin-macro re-initializes `\cals@current@cs` and makes that the left table frame border is of the correct width. The end-macro creates the right border for the right table frame. The most work is performed in the nextcell-macro. Arguments:

1. Width of the cell
2. Width of the left border
3. Width of the right border, *must be a macro name*
4. Background color of the cell

For the special conditions (`\relax`, 0pt, empty name) see the description of `\cals@cs@outOne`. The right border is not typeset immediately. Instead, it is saved to `\cals@lastWidth` (as `\relax` if no overrides) and is handled by the next call to `nextcell`.

```
\newcommand\cals@csrow@begin{%
\setbox\cals@current@cs=\box\voidb@x %
\let\cals@lastWidth=\relax}

\newcommand\cals@csrow@nextcell[4]{%
```

For the first cell, temporarily re-define the default border width to the frame border. Macro `\next` will restore it back.

```
\ifvoid\cals@current@cs
  \toks0=\expandafter{\cals@cs@width}%
  \def\next{\edef\cals@cs@width{\the\toks0}}%
  \edef\cals@cs@width{\cals@framecs@width}%
\else \let\next=\relax \fi
```

Create the decorations, remember the right border. Restore the value `lastLeftWidth` after the end of the hbox-group.

```
\cals@maxWidth\cals@lastWidth{#2}%
\setbox\cals@current@cs=\hbox{\unhbox\cals@current@cs
    \cals@cs@outOne{#1}\cals@width{#4}%
    \global\let\cals@tmp=\cals@lastLeftWidth}%
\let\cals@lastLeftWidth=\cals@tmp
\let\cals@lastWidth=#3%
```

Restore the old value of the default border width.

```
\next}

```

User-specified width (initially by `borderR`, now located in `lastWidth`) must override the frame width.

```
\newcommand\cals@csrow@end{%
\ifx \relax\cals@lastWidth
 \let\cals@width=\cals@framecs@width
\else
 \let\cals@width=\cals@lastWidth
\fi
\cals@csrow@nextcell\relax\cals@width\relax\relax}
```

4.5 Row separation (rowsep)

4.5.1 Data presentation

A horizontal line between table rows can't appear in the output immediately. Its formatting should be postponed until the next row and its context are known. Two basic cases:

- The default cell formatting is to have a border around cells. For some cell A in the middle of a table, the user has overridden formatting to no borders.

To imlement the user's wish, we should not create the bottom border for the cell *B* which is above the *A*. But we don't know about the wish for *A* while processing *B*. Therefore, the border between two rows can be established only after processing the both rows.

- The default cell border is 1pt, but a border between a body and a header or a footer row is 2pt. It means that the border between two rows should be created only after we sure that there is no table break.

Our approach is to define the desired formatting of a rowsep as a set of parameters in a token list. Later we can join two rowseps or a rowsep and context to a final rowsep.

A rowsep token list consist of several items, each item is a list of tokens or token groups:

1. length
2. left-border
3. right-border
4. user-specified thickness

Values *left-border* and *right-border* are required to get a nice rectangle border around a cell. Without length correction, using the cell dimension, the border could look like:

```
  xxxxxxxx
xxx cell xxx
  xxxxxxxx
```

With length correction, we get the correct result:

```
xxxxxxxxxxxx
xxx cell xxx
xxxxxxxxxxxx
```

Here is an example of a rowsep specification. It consist of three items. The first item: length is 5cm, width 2pt, borders are 2mm. The second item: length is 9cm, borders are 2mm, no rule at all. The third item: length 2cm, default width, borders are 2pt. The token list for this specification:

```
{ {5cm} {2mm} {2mm} {2pt} }
{ {9cm} {2mm} {2mm} {0pt} }
{ {2cm} {2pt} {2pt} \relax }
```

Cell length and the left and right borders should be the correct lengths, the rule thickness can be `\relax`, in this case the actual thickness will be calculated during output.

`\cals@rs@pack` Construct a rowsep fragment from the arguments 2-5 and put it to the macro 1.

```
\newcommand\cals@rs@pack[5]{%
\edef#1{\noexpand{#2\noexpand}\noexpand{#3\noexpand}\noexpand{#4\noexpand}%
 \ifx \relax#5\relax \else \noexpand{#5\noexpand}\fi }}
```

`\cals@rs@unpack` The reverse for `\cals@rs@pack`. The first argument is a rowsep fragment (without enclosing curly braces), arguments 2-5 is macro names where to put the results.

```
\newcommand\cals@rs@unpack[5]{%
\def\cals@tmp##1##2##3##4{\edef#2{##1}\edef#3{##2}\edef#4{##3}%
 \ifx\relax##4\let#5=\relax \else \edef#5{##4}\fi}%
\expandafter\cals@tmp#1}
```

4.5.2 From individual decorations to rowsep specification

The rowsep specifications are created cell-by-cell and stored in the macros `\cals@current@rs@above` and `\cals@current@rs@below`. The construction happens with delay because we don't know the exact value of the right border until the next cell is processed.

`\cals@rs@spec@begin` Initializes `\cals@current@rs@above`, `\cals@current@rs@below` and set the flag of a new row.

```
\newcommand\cals@rs@spec@begin{%
\def\cals@current@rs@above{}%
\def\cals@current@rs@below{}%
\let\cals@rs@spec@ll=\relax}
```

`\cals@rs@spec@next`
`\cals@rs@spec@nextII` Finalizes the decorations for the previous cell by using the left border of the current as the right border for the previous. Then remembers the decorations of the current cell — the left border width, the widths of the top and bottom borders (`\relax` is ok) — in the macros `\cals@rs@spec@ll`, `...@bl`, `...@bt`, `...@bb`. All the arguments much be macros.

```
\newcommand\cals@rs@spec@next[4]{
\cals@rs@spec@nextII#2
\let\cals@rs@spec@ll=#1%
\let\cals@rs@spec@bl=#2%
\let\cals@rs@spec@bt=#3%
\let\cals@rs@spec@bb=#4%
}

\newcommand\cals@rs@spec@nextII[1]{%
\ifx \relax\cals@rs@spec@ll \else
 \cals@rs@pack\cals@tmp\cals@rs@spec@ll\cals@rs@spec@bl#1\cals@rs@spec@bt
 \llt@snoc\cals@current@rs@above\cals@tmp
 \cals@rs@pack\cals@tmp\cals@rs@spec@ll\cals@rs@spec@bl#1\cals@rs@spec@bb
 \llt@snoc\cals@current@rs@below\cals@tmp
\fi
}
```

`\cals@rs@spec@end`

Finishes the rowsep specification by putting the last cell to it. The only implicit argument (`\cals@lastLeftWidth`) is the width of the right border of the last cell.

```
\newcommand\cals@rs@spec@end[1]{}
\let\cals@rs@spec@end=\cals@rs@spec@nextII
```

4.5.3 “Waiting” rowsep

`\cals@rs@sofar@length`
`\cals@rs@sofar@borderl`
`\cals@rs@sofar@borderr`
`\cals@rs@sofar@width`

Typesetting a row separator is not an easy task, especially because we support border-widths. Indeed, consider the worst case: four cells and all the borders are different. Our solution is an optimizer for a good case. We do not typeset a fragment of the rule immediately. Instead, we remember the parameters. If the next fragment is of the same width, we increase the length of the “waiting” fragment. Otherwise, we output the waiting fragment and the new fragment becomes the new waiting fragment.

```
\newcommand\cals@rs@sofar@length{}
\newcommand\cals@rs@sofar@borderl{}
\newcommand\cals@rs@sofar@borderr{}
\newcommand\cals@rs@sofar@width{}
```

`\cals@rs@sofar@reset`

Sets a flag that a new waiting rule should be started.

```
\newcommand\cals@rs@sofar@reset{\let\cals@rs@sofar@width=\relax}
```

`\cals@rs@sofar@end`

Prints the waiting rule, if exists.

```
\newcommand\cals@rs@sofar@end{\ifx\relax\cals@rs@sofar@width
 \else\cals@rs@sofar@out\fi}
```

`\cals@rs@sofar@next`

Enlarges the current waiting rule, or typesets it and starts new if the widths do not match. All the parameters must be macro names. In order: length, left border, right border, width.

```
\newcommand\cals@rs@sofar@next[4]{%
\ifx\relax\cals@rs@sofar@width
```

Starts a new waiting rule.

```
 \let\cals@rs@sofar@length=#1%
 \let\cals@rs@sofar@borderl=#2%
 \let\cals@rs@sofar@borderr=#3%
 \let\cals@rs@sofar@width=#4%
\else
 \ifdim \cals@rs@sofar@width=#4\relax
```

Enlarges the waiting rule.

```
  \dimen0=\cals@rs@sofar@length\relax
  \advance\dimen0 by #1\relax
  \edef\cals@rs@sofar@length{\the\dimen0}%
  \let\cals@rs@sofar@borderr=#3%
 \else
```

Typesets the current and start a new waiting rule.

```
  \cals@rs@sofar@out
  \let\cals@rs@sofar@length=#1%
  \let\cals@rs@sofar@borderl=#2%
  \let\cals@rs@sofar@borderr=#3%
  \let\cals@rs@sofar@width=#4%
 \fi
\fi}
```

`\cals@rs@sofar@over` Repeats the last rowsep fragment, probably with another settings. Arguments are like in `\cals@rs@sofar@next`.

```
\newcommand\cals@rs@sofar@over[4]{%
\ifdim 0pt=#4
  \relax
\else
  \ifdim \cals@rs@sofar@width=#4\relax
```

The width is not changed. We probably need to enlarge the right border and probably the left border too. The latter is a bit harder because we don't want to change it if the line continues from another cell (so, change only if `length`+`borderl`>`sofar@length`+`sofar@borderl`).

```
    \ifdim #3>\cals@rs@sofar@borderr\relax
      \edef\cals@rs@sofar@borderr{#3}%
    \fi
    \dimen0=\cals@rs@sofar@length
    \advance\dimen0 by \cals@rs@sofar@borderl\relax
    \advance\dimen0 by -#2\relax
    \ifdim #1>\dimen0 \relax
      \edef\cals@rs@sofar@borderl{#2}%
    \fi
  \else
```

Typesets the current and start a new waiting rule.

```
    \cals@rs@sofar@out
    \hskip-#1\relax
    \let\cals@rs@sofar@length=#1%
    \let\cals@rs@sofar@borderl=#2%
    \let\cals@rs@sofar@borderr=#3%
    \let\cals@rs@sofar@width=#4%
  \fi
\fi}
```

`\cals@rs@sofar@out` Typesets the waiting rule

```
\newcommand\cals@rs@sofar@out{%
\ifdim 0pt=\cals@rs@sofar@width\relax
  \hskip\cals@rs@sofar@length\relax
\else
  \cals@halfWidthToDimen0\cals@rs@sofar@borderl
  \hskip-\dimen0\relax
  \cals@halfWidthToDimen2\cals@rs@sofar@borderr
  \dimen4=\cals@rs@sofar@length\relax
  \advance\dimen4 by \dimen0\relax \advance\dimen4 by \dimen2\relax
  \cals@halfWidthToDimen6\cals@rs@sofar@width
  \vrule height\dimen6 depth\dimen6 width\dimen4\relax
  \hskip-\dimen2\relax
\fi}
```

4.5.4 From rowsep specification to typesetting

`\cals@rs@joinTwo` Join and typeset two rowseps (arguments 2 and 3, must be macro names). The number and the lengths of the fragments in the rowseps should match. The argument 1 (also a macro name) is the default width. Corrupts the macros 2 and 3. Call this macro inside `sofar@reset`...`@end` group.

```
\newcommand\cals@rs@joinTwo[3]{%
```

The loop function.

```
\def\next##1{%
\ifx \eol##1\let\next=\relax
\else
  \toks0=\expandafter{##1}%
  \edef\cals@tmpII{\the\toks0}%
  \llt@decons#3%
```

Now `\cals@tmpII` contains a current fragment of the first rowsep, and `\llt@car` of the second. Unpack the individual parameters.

```
  \cals@rs@unpack\cals@tmpII\cals@tmpLI \cals@tmpBlI \cals@tmpBrI \cals@tmpWI
  \cals@rs@unpack\llt@car   \cals@tmpLII\cals@tmpBlII\cals@tmpBrII\cals@tmpWII
```

The special case is when we should not typeset a rowsep fragment.

```
  \let\cals@tmp=\cals@iftrue
  \cals@maxWidth\cals@tmpWI\cals@tmpWII
  \ifx \relax\cals@width\else \ifdim \cals@width=0pt %
    \cals@rs@sofar@next\cals@tmpLI\cals@tmpBlI\cals@tmpBrI\cals@width
    \let\cals@tmp=\cals@iffalse
  \fi\fi
```

Not the special case. Put the both definitions, and let the underlying functions take care of calculations.

```
  \cals@tmp\ifvoid
    \cals@widthII#1\cals@tmpWI
    \cals@rs@sofar@next\cals@tmpLI\cals@tmpBlI\cals@tmpBrI\cals@width
    \cals@widthII#1\cals@tmpWII
    \cals@rs@sofar@over\cals@tmpLII\cals@tmpBlII\cals@tmpBrII\cals@width
  \fi
\fi
```

End of `\next` definition: continue the loop. End of `\cals@rs@joinTwo` definition: start the loop.

```
\next}%
\expandafter\next#2\eol}
```

`\cals@rs@joinOne` A simplified version of the previous macro. We have only one rowsep, and want to join and typeset it with regard to some width, given as the first macro parameter. Call this macro inside `sofar@reset`...`@end` group.

```
\newcommand\cals@rs@joinOne[2]{%
\def\next##1{\ifx\eol##1\let\next=\relax\else
 \toks0=\expandafter{##1}%
 \edef\cals@tmpII{\the\toks0}%
 \cals@rs@unpack\cals@tmpII\cals@tmpL\cals@tmpBl\cals@tmpBr\cals@tmpW
 \cals@widthII#1\cals@tmpW
 \cals@rs@sofar@next\cals@tmpL\cals@tmpBl\cals@tmpBr\cals@width
\fi\next}%
\expandafter\next#2\eol}
```

4.6 RTL (right-to-left) hooks

`\if@RTL`
`\if@RTLtab`
`\@RTLtabtrue`

Provide RTL status commands even if the RTL packages are not loaded.

```
\def\next{%
  \let\if@RTL=\iffalse
  \let\if@RTLtab=\iffalse
  \let\@RTLtabtrue=\relax
}
\ifdefined\if@RTL \relax \else \next \fi
```

`\cals@setup@alignment` Swap alignment in the RTL mode.

```
\newcommand\cals@setup@alignment[1]{%
\if c#1\relax \cals@vfillAdd \leftskip \cals@vfillAdd \rightskip \fi
\if@RTL
 \if l#1\relax \cals@vfillAdd \leftskip \cals@vfillDrop\rightskip \fi
 \if r#1\relax \cals@vfillDrop\leftskip \cals@vfillDrop\rightskip \fi
\else
 \if l#1\relax \cals@vfillDrop\leftskip \cals@vfillDrop\rightskip \fi
 \if r#1\relax \cals@vfillAdd \leftskip \cals@vfillDrop\rightskip \fi
\fi
}
```

`\cals@hskip@lr` Do `hskip` with the first argument, unless in the RTL mode.

```
\newcommand\cals@hskip@lr[2]{%
\if@RTL \hskip#2\relax \else \hskip#1\relax \fi}
```