The cals package*

Oleg Parashchenko
olpa@uucode.com

October 14, 2015

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, pdfsync 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`

The most important feature: the table (its rows) must start in a vertical mode, the cells content should switch to a horizontal mode.

Please post questions and suggestions to TeX-SX (`http://tex.stackexchange.com/`), the newsgroup `comp.text.tex` and the `texhax` mailing list (see `http://tug.org/mailman/listinfo/texhax`), not directly to me.

Summary of the user interface:

```
\begin{calstable}
\colwidths{{100pt}{200pt}}
\brow \cell{a} \cell{b} \erow
\end{calstable}
```

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

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`.

*This document corresponds to cals CALS, dated 2015/10/14.

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
```