The cals package: Multipage tables with decorations

Oleg Parashchenko

Abstract

Tables are one of the most complicated parts of any typesetting or publishing system, and LaTeX is no exception. There are a number of packages related to tables, but so far the following goal has been unreachable: to automatically typeset huge, complex and attractive tables.

The new package cals makes this possible.

1 Introduction

I use TeX as an alternative to XSL-FO [2] for publishing XML as PDF. The customers do not care about LaTeX restrictions and guidelines (for example, "never use vertical rules" from booktabs [4]); they demand their specified layout. I failed to implement their complex requirements for tables using existing LaTeX packages and decided to write my own. The name "cals" comes from "CALS Table Model" [1], a standard for table markup in XML.

The key features are:

- huge tables
- spanned cells
- decorations
- automatic typesetting

Different table packages implement different approaches to break a table across pages; see the TeX FAQ [8], "Tables longer then a single page". The cals package typesets the current row in memory, checks if the rest of the page is enough for the row, forces a page break if required, and finally flushes the row. This way, only a bit of memory is required, and therefore tables can be long. As a downside, the widths of columns need to be provided by the user.

The TeX code for tables is supposed to be generated automatically, therefore the syntax is not traditional and maybe not convenient for manual coding. Instead of dividing a row into cells using `&`, each cell is introduced by a named command.

The implementation of decorations is unique throughout TeX, to my knowledge. Table rules are:

- style-driven and
- context-sensitive.

A stylesheet defines how a typical table looks. The user need only give cells; the decorations do or do not appear automatically. The width of a border depends on its location; it is different for the table frame and for the header separator line.

The cals package has many features, but some unusual requirements might not be supported. If you want to make changes or just look at the implementation, the source code and support files are available at `http://github.com/olpa/tex/` in the directory `cals`.

2 User's guide

This section

- provides a summary of the cals commands,
- shows how to use the commands, and
- suggests compatibility strategies.

A complete document `demo.tex` (`demo.pdf` [6], also on CTAN) from the package documentation contains examples of:

- a simple table,
- decoration control,
- cell spanning, and
- a multipage table inside a multicols environment inside a table.

2.1 Summary

First, an overview of cals commands. Details and examples follow.

Table elements:

```
\thead, \tfoot
\tbreak{\penalty-10000}
\lastrule
```

Alignment:

```
\alignL, \alignC, \alignR
\vfill
```

Padding (lengths):

```
\cals@setpadding{Ag}
\cals@paddingL,  \cals@paddingT
\cals@paddingR,  \cals@paddingB
\cals@setcellprevdepth{Al}
\cals@paddingD
```

Color:

```
\cals@bgcolor
```

Rules (macros as lengths):

```
\cals@cs@width, \cals@rs@width
\cals@framecs@width, \cals@framers@width
\cals@bodyrs@width
\cals@borderL,  \cals@borderT
\cals@borderR,  \cals@borderB
```

Hooks:

```
\cals@AtBeginCell, \cals@AtEndCell
```

Spanning:

```
\nullcell
\spancontent
```

2.2 Simple tables

Sample code:

```
\par
\begin{calstable}
\colwidths{{50pt}{100pt}}
\brow  \cell{a} \cell{b}  \erow
\brow  \cell{c} \cell{d}  \erow
\end{calstable}
```

a	b
c	d

Basics:

- Tables are created with the environment `calstable`.
- Column widths must be specified explicitly.
- Each row is marked by the `\brow...\erow` pair.
- Individual cells are specified by the command `\cell`.

And more specifics:

- Tables must start in vertical mode.
- Cells are vboxes, i.e., TEX uses restricted vertical mode to typeset the content.
- Changes inside `\cell{...}` are local.
- The macros `\cals@AtBeginCell` and `\cals@AtEndCell` are called at the boundaries of a cell group.
- The pair `\brow...\erow` does *not* make an implicit group. All changes are active till the end of the table.

2.3 Multipage tables

Cals tables are split over pages automatically. Such tables benefit from repeatable headers and footers, specified by the commands `\thead` and `\tfoot`.

```
\begin{calstable}
\colwidths{{50pt}{100pt}}
%
\thead{\bfseries\selectfont
  \brow \cell{col1} \cell{col2} \erow
  \mdseries\selectfont}
\tfoot{\lastrule\nointerlineskip
  \textit{\strut Some table caption
    (not implemented: PartKofN)}\par}
%
\brow \cell{r1,col1} \cell{r1,col2} \erow
\brow \cell{r2,col1} \cell{r2,col2} \erow
\brow \cell{r3,col1} \cell{r3,col2} \erow
\tbreak{Manual table break!\strut\par}
\brow \cell{r4,col1} \cell{r4,col2} \erow
\brow \cell{r5,col1} \cell{r5,col2} \erow
...1000 rows...
\end{calstable}
```

col1	**col2**
r1,col1	r1,col2
r2,col1	r2,col2
r3,col1	r3,col2

Some table caption (not implemented: PartKofN)

Manual table break!

col1	**col2**
r4,col1	r4,col2
r5,col1	r5,col2

... 1000 rows ...

Some table caption (not implemented: PartKofN)

Comments:

- `\thead` and `\tfoot` must be given before the table body.
- Small distraction: text is bold in the header and is reset before the body starts.
- `\thead` and `\tfoot` can contain any vertical material. In such a case, `\tfoot` should use the command `\lastrule` where the table ends, so the code decorates the table correctly.
- I'd like to implement "Part K of N" functionality, but can't say when it will happen.

As long as the current row plus the footer fits on the rest of the page, there is no page break. Otherwise, cals emits the footer, page break, the header, and only then the current row.

A manual table break can be made using the command `\tbreak{`⟨*smth*⟩`}`, where ⟨*smth*⟩ is what to emit between the footer and the next header. Most likely, it is `\vfill\break`.

2.4 Alignment

To left, center, or right-align the content of a cell, use `\alignL`, `\alignC` or `\alignR`, respectively. The default is left-alignment. To vertically align a cell to the middle or bottom, add `\vfil` or `\vfill` before the cell content.

```
\begin{calstable}
\colwidths{{60pt}{60pt}{60pt}}
\def\cals@framers@width{0.4pt}
\def\cals@framecs@width{0.4pt}
%
\brow
 \alignR \cell{\vfill right, bottom}
 \alignC \cell{\vfil center, middle}
 \alignL \cell{left, top}
\ht\cals@current@row=50pt
\erow
\end{calstable}
```

right, bottom	center, middle	left, top

For demonstration purposes, the example sets the height of rows. This is an undocumented and unplanned feature. You should not rely on it.

2.5 Padding

Padding depends on the current font and is calculated when a table starts. If you change a font inside a table, it is a good idea to update the padding:

```
\cals@setpadding{Ag}
\cals@setcellprevdepth{Al}
```

The first command sets the left, top, right and bottom padding: `\cals@paddingL`, `\cals@paddingT`, `\cals@paddingR`, and `\cals@paddingB`, respectively. The value is half of the height of the argument. The second command sets the length `\cals@paddingD`, which helps to align baselines in a row. More details on the 'Ag' and 'Al' are discussed later.

```
\fontsize{20pt}{22pt}\selectfont
...
\begin{calstable}
\colwidths{{70pt}{70pt}{70pt}}
%
\fontsize{10pt}{12pt}\selectfont
\brow
  \cell{Padding} \cell{is too} \cell{big}
\erow
%
\cals@setpadding{Ag}
\cals@setcellprevdepth{Al}
\brow
  \cell{This} \cell{padding}
  \cell{is better}
\erow
%
\setlength{\cals@paddingT}{0pt}
\setlength{\cals@paddingB}{0pt}
\brow
  \cell{Zero padding} \cell{aaaaaa}
  \setlength{\cals@paddingD}{-10000pt}
  \cell{aaaaaa}
\erow
\end{calstable}
```

Padding	is too	big
This	padding	is better
Zero padding	aaaaaa	aaaaaa

In this example, we make the table font smaller then the document font. In the first row, the padding is too big. Then the padding is updated, and the second row looks better. In the third row, the top and bottom padding are set to zero. However, the second cell has additional space at top to align the baseline. To omit this, the length `\cals@paddingD` is disabled in the third cell.

2.6 Colors and rules

Specifying a color is straightforward: if the macro `\cals@bgcolor` is non-empty, its value is the name of the cell color.

The width of a cell border (rule) depends on the context:

- The usual borders get widths from the macros `\cals@cs@width` and `\cals@rs@width`.
- The table frame uses `\cals@framecs@width` and `\cals@framers@width`.
- The separation between the table body and its header or footer is `\cals@bodyrs@width`.

The default settings are correspondingly `0.4pt` (the usual line for usual cells), `0pt` (table frame is absent) and `1.2pt` (header and footer are delimited by a thick line). All the borders are "phantoms" and do not affect layout.

Border types are further divided into subtypes: `cs` means "column separation" (left and right borders), and `rs` means "row separation" (top and bottom borders).

Finally, there are overrides. If any of the macros `\cals@borderL`, `\cals@borderT`, `\cals@borderR`, or `\cals@borderB` are defined, they specify the width of the left, top, right or bottom border, ignoring the cell's context. By default, these macros are assigned `\relax` and are thus inactive.

```
\begin{calstable}
\colwidths{{60pt}{60pt}{60pt}}
\def\cals@cs@width{1pt}
\def\cals@rs@width{0pt}
\def\cals@framers@width{2pt}
\def\cals@framecs@width{1pt}
% background swap
\def\c{\ifx\cals@bgcolor\empty
  \def\cals@bgcolor{lightgray}
  \else \def\cals@bgcolor{} \fi}
%
\brow \c\cell{A}
  \c\cell{B} \c\cell{C} \erow
%
\brow \c\cell{D}
 \def\cals@borderL{3pt}
 \def\cals@borderT{4pt}
```

```
 \def\cals@borderR{5pt}
 \def\cals@borderB{6pt}
 \c\cell{E}
 \let\cals@borderL=\relax
 \let\cals@borderT=\relax
 \let\cals@borderR=\relax
 \let\cals@borderB=\relax
\c\cell{F} \erow
%
\brow \c\cell{G}
  \c\cell{H} \c\cell{I} \erow
\end{calstable}
```

A	B	C
D	E	F
G	H	I

In this example, the macro `\c` alternately sets and disables a color of a cell.

2.7 Spanned cells

To define a spanning area, use the `\nullcell` command for each component cell. The argument of this command specifies the location of the cell: `l` if on the left edge, `t` on the top, `r` on the right and `b` on the bottom. To typeset the spanning area, use the command `\spancontent`, which should be given immediately after right-bottom component cell.

The following table illustrates the words, providing an example how to typeset three spanned areas of different shapes.

```
\let\nc=\nullcell
\let\sc=\spancontent
```

\nc{ltr}	\nc{ltb}	\nc{tb}	\nc{tbr} \sc{...}
\nc{lr}	\nc{lt}	\nc{t}	\nc{tr}
\nc{lr}	\nc{l}	\nc{}	\nc{r}
\nc{lbr} \sc{...}	\nc{lb}	\nc{b}	\nc{br} \sc{...}

As an example, here is a "spiral" in a 3×3 table.

```
\begin{calstable}
\colwidths{{40pt}{40pt}{40pt}}
\def\cals@framecs@width{0.4pt}
\def\cals@framers@width{0.4pt}
\brow
 \nullcell{ltr}
 \nullcell{ltb}
 \nullcell{trb}\spancontent{b3, c3}
 \ht\cals@current@row=40pt
\erow
\brow
 \nullcell{lbr}\spancontent{a2, a3}
 \cell{b2}
 \nullcell{ltr}
 \ht\cals@current@row=40pt
\erow
\brow
 \nullcell{ltb}
 \nullcell{btr}\spancontent{a1, b1}
 \nullcell{blr}\spancontent{c1, c2}
 \ht\cals@current@row=40pt
\erow
\end{calstable}
```

a2, a3	b3, c3	
	b2	c1, c2
a1, b1		

2.8 User-level tricks

There are a few compatibility issues and out-of-design uses. So far, they are:

- pdfsync compatibility,
- multicol compatibility,
- inter-row page breaks.

2.8.1 pdfsync support

The package pdfsync seems obsolete, but is still in use. It registers `\every`-hooks and inserts synchronization markers. The cals package does not expect such interference and fails. The solution is:

- disable pdfsync inside a table,
- temporarily enable it inside a cell.

Sample code:

```
\makeatletter
\let\oldcalstable=\calstable
\def\calstable{\oldcalstable\pdfsyncstop}
\def\cals@AtBeginCell{\pdfsyncstart}
```

We use `\def` instead of `\let` for `\cals@AtBeginCell` because `\pdfsyncstart` and `\pdfsyncend` do not exist in the preamble; they are defined during execution of `\begin{document}`.

2.8.2 multicols compatibility

If a cell contains a multicols environment, the content is not padded. This is a side effect of the technical implementation:

- padding is implemented using `\leftskip`,
- multicols issues boxes in vertical mode. In this case, TEX ignores `\leftskip`.

The solution is easy and quite unexpected: we pretend that the cell is a list item:

```
\cell{
\makeatletter
\@totalleftmargin=\cals@paddingL\relax
\begin{multicols}{2}
... now ok ...
\end{multicols}
}
```

2.8.3 Inter-row page breaks

All the typesetting systems I have seen break long tables between rows. But if rows are very tall, it may be useful to break even within a row, to make full use of the page height.

The cals package can be extended to support inter-row breaks. A proof of concept for a simplified case (no spanned cells, no vertical alignment) is published in `comp.text.tex` [7].

A complete solution is not presently available. The hardest part is to code a generic `\vsplit` — dividing a box and its internal sub-boxes into two parts, "before" and "after". Contributions are welcome.

3 Technical details

For advanced customization and adding new functionality, one needs to understand the internals of the cals package. The rest of the article requires advanced TEX and LATEX knowledge.

3.1 Padding and alignment of cells

In short, a cell is a vbox.

3.1.1 Horizontal

Horizontal dimensions are applied indirectly when the cell content switches TEX to the restricted horizontal mode. Before typesetting the content, the parameters are set:

- `\hsize` = width of the cell
- `\leftskip` = left padding
- `\rightskip` = right padding

If a mode switch does not occur (for example, the content is wrapped by a box), then TEX does not use `\leftskip` and `\rightskip`, and you get no left and right padding. This is not a bug, this is a feature. If the automatic width of a box is incorrect, cals handles this case and forces the right value (`\wd\boxX=YY`).

Horizontal alignment is also implemented using `\leftskip` and `\rightskip`. The command `\alignC` adds `plus1fill` to both skips, `\alignR` only to the right one, and `\alignL` drops the plus/minus components.

3.1.2 Vertical

Before closing the box, cals adds:

```
\vfil \vskip\cals@paddingB
```

This code aligns the content top and sets the bottom padding.

The start of a cell is more complicated:

```
\vskip -⟨rowspan_ compensation⟩
\vskip\cals@paddingT
\vskip-\parskip
\prevdepth=\cals@paddingD
```

In addition to the top padding (`\cals@paddingT`), there are a number of adjustments. The first is conditional and happens only if cals typesets a row-spanned cell. In this case, the negative skip increases the visual height of the cell, so visually the cell starts on the first row of spanning.

On switching from a vertical to a horizontal mode, TEX adds `\parskip` glue, but we do not need this at the beginning of a cell; therefore we annihilate it. Finally, there is `\baselineskip` glue, implicitly set by `\prevdepth`. The value is calculated in such a way, that the distance between the top border and the top of the letters "Al" is exactly `\cals@paddingT`.

Cell content is packed vertically twice. First, as a normal vbox. Second, after the final height of its row is known, the cell is unvboxed and put into a vbox of the target height.

3.2 Decorations

A table row and its decorations are separated. To illustrate the explanations, I use the second row from the following sample table (normal border is 1pt, first column right border 2pt, second column right border 3pt, right table frame 4pt, thick horizontal border 5pt, columns are 60pt, 70pt, 80pt):

a3	b3	c3
a2	b2	c2
a1	b1	c1

At some point after `\erow`, when `\cals@issue@row` is called, the following boxes and macros are set:

- `\cals@current@row`
- `\cals@current@cs`
- `\cals@current@rs@above` and `\cals@current@rs@below`
- `\cals@last@rs@below`

The row content resides in `\cals@current@row`:

a2 b2 c2

Here is an annotated dump:

```
> \box\cals@current@row=
\hbox(15.72218+0.0)x210.0 % Second row
```

```
.\vbox(15.72218+0.0)x60.0 % The cell "a2"
..\glue 4.38887           % \cals@paddingT
..\glue 0.0 plus -1.0     % -\parskip
..\glue(\parskip) 0.0 plus 1.0
..\glue(\baselineskip) 0.5
..\hbox(6.44444+0.0)x60.0, glue set 41fil
...\glue(\leftskip) 4.388 % \cals@paddingL
...\hbox(0.0+0.0)x0.0
...\OT1/cmr/m/n/10 a
...\OT1/cmr/m/n/10 2
...\penalty 10000
...\glue(\parfillskip) 0.0 plus 1.0fil
...\glue(\rightskip) 4.38 % \cals@paddingR
..\glue 0.0 plus 1.0fil   % \vfil
..\glue 4.38887           % \cals@paddingB
.\vbox(15.72218+0.0)x70.0 % The cell "b2"
..  ...b2 here...
.\vbox(15.72218+0.0)x80.0 % The cell "c2"
..  ...c2 here...
```

This dump illustrates also the structure of a cell, with all the glue items, as described in the previous section.

The width of a border is defined by its location, unless the user explicitly sets the width. If the user sets different widths for a common border of adjoined cells, the greater value wins. (This is related to why cals does not support border colors or styles: I have no idea what to do if one cell wants, for example, green border and another red.)

Background color and vertical borders are in the box `\cals@current@cs`:

Horizontal rules are not yet typeset. Instead, they are described by the macros `\cals@current@rs@above` (row separation above the row) and the analogous `\cals@current@rs@below`, as follows:

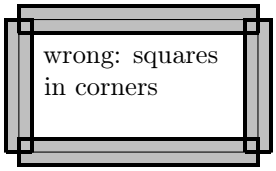
```
> \cals@current@rs@above=macro:
->{{60pt}{1pt}{2pt}\relax }
  {{70pt}{2pt}{3pt}{5pt}  }
  {{80pt}{3pt}{4pt}\relax }.
> \cals@current@rs@below=macro:
->{{60pt}{1pt}{2pt}\relax }
  {{70pt}{2pt}{3pt}\relax }
  {{80pt}{3pt}{4pt}\relax }.
```

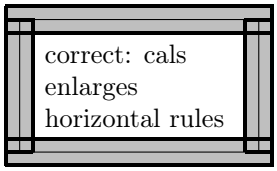
The macros consists of N groups of four, where N is the number of columns. Each record contains:

- the length of the rule fragment (the width of the column)
- the width of the left border
- the width of the right border
- the width of the rule or `\relax`

Unless the user manually sets the width using `\cals@borderT` or `\cals@borderB`, the last field contains `\relax`. It means "as yet unknown", and the code will decide on the width later, when the location of the rule is clear.

The borders should be a bit longer then cell dimensions, in order to create a closed frame instead of leaving empty squares in the corners.





The procedure of typesetting a rule combines three components: 1) the default width; 2) the preceding row bottom rule description; 3) the following row top rule description. The code tries to produce as large a rule as possible. Instead of emitting a rule fragment immediately, it remembers the length and width. If the next fragment is the same width, the recorded length is extended. If not, then the pending rule is typeset, and the new fragment is remembered.

Let's trace a simplified version of the algorithm for the separation between the first and the second rows in the sample table. The border width is 1pt, the value of `\cals@current@rs@above` is shown above, and `\cals@last@rs@below` is the same as `\cals@current@rs@below`.

- a3 border. Length 60pt, width 1pt.
  → Remember: L=60pt, W=1pt.
- a2 border. Left 60pt, length 60pt, width 1pt.
  → Nothing changed.
- b3 border. Length 70pt, width 1pt.
  → Increase: L=130pt, W=1pt.
- b2 border. Left 70pt, length 70pt, width 5pt.
  → Emit rule 130x1pt. Emit left skip 70pt.
  Remember: L=70pt, W=5pt.
- c3 border. Length 80pt, width 1pt.
  → Emit rule 70x5pt. Remember: L=80pt, W=1pt.
- c2 border. Left 80pt, length 80pt, width 1pt.
  → Nothing changed.
- End of the row. → Emit rule 80x1pt.

The real procedure is a bit more complicated because it takes into account the corrections for the vertical borders. Still, the real result is very similar to the simplified version:

```
\glue -0.5    % column 1, left border
\rule(0.5+0.5)x132.0 % columns 1 and 2
\glue -1.5    % column 2, right border
\glue -70.0   % back
\glue -1.0    % column 2, left border
\rule(2.5+2.5)x72.5     % column 2
```

```
\glue -1.5    % column 2, right border
\glue -1.5    % column 3, left border
\rule(0.5+0.5)x83.5     % column 3
\glue -2.0    % column 3, right border
```

The first rule is for the first and the second column, the second for a thick border in the second column, and the third rule for the third column. Note that the border in the second column consists of two overlapping rules. Such intersection is not nice, but it happens only when the width is changing, which should not happen often. In the usual case, when the width is constant, the code emits only one rule.

3.3 Spanning

I made several attempts before the current implementation of spanning was developed. In my opinion, the interface balances clarity for users and ease and maintenance of coding.

The command `\nullcell` performs two main tasks:

- calculate the dimensions of the spanned area, `\cals@span@width` and `\cals@span@height`,
- handle decorations.

Calculation of the width of a spanned area starts when the left-bottom `\nullcell` appears (the argument contains both `l` and `b`). The width is updated while the bottom `\nullcell`s continue to appear (the argument contains `b` without `l`).

The height calculation is similar: start in the left-top (both `l` and `t`) `\nullcell`, update in a left (`l` without `t`) one. However, there is a complication in that several spanning areas can interfere. To allow several spans at once, each right (`r` without `t`) `\nullcell` adds the span height to the end of the queue `\cals@spanq@heights`, and each left (`l` without `t`) `\nullcell` takes the saved height from the beginning of the queue. With several active spans, the queue works like a cyclic buffer. The values rise from the end to the beginning, and magically the first value is always the saved height for the coming span.

The decorations of a spanned area are composed from independent parts. More precisely, the command `\spancontent` does not produce decorations at all. Instead, each `\nullcell` works like a usual `\cell` after some tuning. First, it temporarily disables all the borders. Then it looks at the location and restores the corresponding areas. For example, if the `\nullcell` is on the left edge (the argument contains `l`), the settings for the left border are restored. After the decorations are produced, all the border settings are reverted to their original values.

3.4 Multipage

Compared to the other parts of cals, the multipage functionality was very easy to code. On the other hand, it required understanding of how TEX works underneath, which is not my strength. Therefore, I expect bugs in multipage functionality, especially in:

- detecting if a table break is required (the macro `\cals@ifbreak`),
- executing a break within a table (the macro `\cals@issue@break` and the end of the macro `\cals@row@dispatch@nospan`)

Initially I tried to implement multipage tables in an output procedure, which, if a page break occurred within a table, added a footer before the break and a header after the break. It more or less worked, but it could not support decorations: the thickness of a row separator depends on its context. In the output procedure, it is very hard (if possible at all) to remove the old in-body separation and insert a table frame rule. My conclusion is that the main code should know if a table break is expected before typesetting a row, and create the break if required.

The following heuristic seems good: does the current row plus the footer fit into the rest of the page? If it does not, a break is required. There are also a few special cases:

- a break is forced if the user defined the macro `\cals@tbreak@tokens` (using `\tbreak`).
- no break in the header, in the footer, after the header or after the first row of a table chunk.

After a row is finished and its decorations are prepared, cals runs the macro `\cals@row@dispatch`. Its main parameters are:

- the bottom decoration of the previous row (given in `\cals@last@rs@below`) and its context (in `\cals@last@context`); details in the package documentation.
- the current row (`\cals@current@row`), its context (`\cals@current@context`), and decorations (`\cals@current@cs`, `\cals@current@rs@above`, `\cals@current@rs@below`).

Depending on whether there is an active rowspan, there are two different modes of work. First, when a cell spans over rows, the package must avoid a table break between. This is implemented by wrapping the row as a vbox. The row dispatcher emits the row not to the page, but appends it to a temporary box. After the last spanned row is collected, the collection becomes `\cals@current@row` and the collected decorations constitute `\cals@current@row@cs`. Then the dispatcher switches to the normal mode.

In the normal mode, usually it is enough to output the decorations and the row. But if a table break is required, the code saves the current row, typesets the footer, the break, the header and only then emits the saved row.

4 TEX tricks and traps

The code in the cals package contains a few tricks, which can be reused in other tools. There were also a few unexpected problems, which I'd like to discuss here.

Maybe it is time to start collecting "TEX design patterns", as with other programming languages [9].

4.1 Actions after an implicit parameter

A straightforward definition of a command `\cell` could be:

```
\newcommand\cell[1]{%
  ...actions before...
  #1%
  ...actions after... }
```

I disliked this approach because then the macro must collect a potentially big argument, which might degrade performance. Instead, an aftergroup-trick is used to inject post-actions to the token stream. Pseudo-code:

```
\def\cals@cell@end{...actions after...}
\def\cell{...actions before...
 \bgroup\aftergroup\cals@cell@end
 \let\next=% eat '{' of the argument
}%{ Implicit argument follows }
```

This trick is described by Victor Eijkhout in "TEX by Topic" [3], section "12.3.4 `\aftergroup`".

Initially, it was perhaps a premature optimization, but later useful side-effects appeared:

- The changes inside `\cell` are local due to wrapping in a group.
- Verbatim and other special content is supported inside `\cell`. This would be impossible when passing the content as a parameter.

4.2 `\newcommand` for documentation, `\def` for definition

The problem with implicit arguments (as with the `\cell` command in the previous section) is that such macros confuse the readers of the code. It is very easy to overlook that a macro requires more parameters then expected.

To help the reader, cals defines a macro twice, first as a prototype and then as the real thing:

```
\newcommand\cell[1]{}
\def\cell{...macro code...}
```

For me, it is an useful eye-catcher.

4.3 Nested conditions

The following code does not work:

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

The idea is to pre-calculate some condition and use it later. But instead of what the programmer wants — the first `\if` matching the outer `\fi` and `\next` (assigned to `\iftrue` here) matching the inner `\fi` — TEX matches `\if` with the inner `\fi`, and the outer `\fi` causes an error.

As a workaround, cals uses a variant of `\iftrue` and `\iffalse` that drops a following token:

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

The nested condition now looks as:

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

The token `\iftrue` (or any other if-token) after `\next`:

- repairs the if-fi balance,
- is ignored when the code is executed.

4.4 The trap of brace delimiting

When a list of macro parameters ends with "`#{`", it means that the last parameter is delimited by a curly brace:

```
\def\mybox#1#{\hbox#1}
\mybox to 40pt{some text}
```

expands to:

```
\hbox to 40pt{some text}
```

In this code fragment, the macro parameter `#1` is '`to 40pt`'. The surprise is that we can't make it explicit:

```
\mybox{to 40pt}{some text}
```

expands to:

```
\hbox{to 40pt}{some text}
```

The macro parameter is now empty and the box content is "to 40pt". The text "some text" is typeset outside the box. In retrospect and in this simplified example, such expansion is obvious. In the real package, however, I fell into the trap several times.

4.5 The trap of dropped curly braces

To use a macro as a list data type, it is convenient to put list items in groups. For example, a list with items "aaa", "bbb" and "ccc" can be defined as:

```
\def\list{{aaa}{bbb}{ccc}}
```

A straightforward definition of list de-construction on the first element and the rest could be:

```
% wrong
\def\decons@helper#1#2\relax{%
  \def\first{#1}%
  \def\rest{#2}}
\def\decons#1{%
  \expandafter\decons@helper#1\relax}
```

The code works in most cases:

```
\decons\list
\show\first
\show\rest
     =>
> \first=macro:
->aaa.
> \rest=macro:
->{bbb}{ccc}.
```

Unfortunately, two-element lists are de-constructed incorrectly, losing the braces:

```
\def\listII{{aaa}{bbb}}
\decons\listII
\show\first
\show\rest
     =>
> \first=macro:
->aaa.
> \rest=macro:
->bbb.
```

The wrong value of `\rest` is '`bbb`'; the correct result would be '`{bbb}`'. The problem is the second parameter of the macro `\decons@helper`. If the value is '`{{bbb}{ccc}...{xxx}}`' (several items) or empty, the parameter is not changed. But if the value is '`{bbb}`' (exactly one item), TEX interprets the curly braces as a parameter delimiter and drops them. Again, it is not a bug, but an unexpected side effect of TEX's rules.

4.6 Unit testing

The complexity of the cals code is above my TEX skills. Fortunately, automated tests helped to keep the code under control.

There are a number of tools for testing TEX code (for example, qstest [5]), but I decided that it would be faster to write my own framework instead of mastering an existing one. So far, I think this was a good decision.

I wrote my tool in Python. It is straightforward. For each test:

- It takes the code fragment, assembles a complete document and compiles it.
- It extracts LATEX messages and the output of `\show`-commands from the log file and asserts that the result is equal to the known master.
- If the test comes with PNG images, then the tool asserts that the generated PDF, after conversion to PNG, is equal to the master images.

The tests for cals and the tool itself are located in the directory `test` of the cals package. To run the tests, execute:

```
$ export TEXINPUTS=`pwd`/../cals:
$ python support/run_tests.py
```

To run a subset of the tests, simple filtering is supported. For example, to run only the tests with the names containing "cell", execute:

```
$ python support/run_tests.py cell
```

The testing framework is generic and can be used in other projects as well. If there is positive feedback and use by other developers, the tool will be extracted to a separate CTAN package.

◇ Oleg Parashchenko
bitplant.de GmbH, Fabrikstr. 15
89520 Heidenheim, Germany
olpa (at) uucode dot com
http://uucode.com/

References

[1] OASIS. TM 9502:1995 – CALS table model DTD. http://www.oasis-open.org/specs/a502.htm, 2001.

[2] W3C. Extensible stylesheet language (XSL), version 1.0. W3C recommendation 15 Oct 2001. http://www.w3.org/TR/2001/REC-xsl-20011015/, 2001.

[3] Victor Eijkhout. *TEX by Topic: A TEXnician's Reference*. Addison-Wesley, 1992.

[4] Simon Fear. Publication quality tables in LATEX. http://mirror.ctan.org/macros/latex/contrib/booktabs/booktabs.pdf, 2005.

[5] David Kastrup. qstest, a LATEX package for unit tests. *TUGboat*, 29(1):193–198, 2008.

[6] Oleg Parashchenko. CALS tables demo. http://mirror.ctan.org/macros/latex/contrib/cals/examples/demo.pdf, 2011.

[7] Oleg Parashchenko. Re: Multi-page table with inter-row page breaks. comp.text.tex, http://groups.google.com/group/comp.text.tex/msg/8141d45708fe85c2, 6 Dec 2010.

[8] UK TUG. TEX frequently asked questions. http://www.tex.ac.uk/faq, 2011.

[9] Wikipedia. Design pattern (computer science). http://en.wikipedia.org/wiki/Design_pattern_(computer_science), 2011.