The PracTEX Journal, 2010, No. 2
Article revision 2010/10/18

Square, multiline cells using tabular(x)

Bastiaan Jacques

Abstract I describe a method for creating square cells, containing multiple lines typeset in paragraph mode using the `array` package. Both plain LaTeX `tabular` and the `tabularx` packages are facilitated.

1 Introduction

Will Robertson describes how to create square cells in LaTeX. Robertson's approach has one limitation: it does not handle variable cell contents, particularly multiple lines typeset in paragraph mode, especially well.[1]

In this article I describe a solution that permits creation of square cells that (can) contain several lines of text, varying in number.[2] This article only concerns itself with the question of *how* to create such tables, not whether doing so would be a good idea.[3] [4]

Lorem ipsum dolor sit	amet, consectetur ...	elit, sed do eiusmod ...
ut labore et dolore	magna aliqua.	Ut enim
ad minim veniam quis	nostrud exercitation	ullamco laboris

Figure 1: The target

2 Struts

Let's start with a little background. The approach most frequently taken to increase cell height is to use struts. Here is a

1. It seems likely that this use was never intended by Robertson; this comment should not be taken as a criticism.
2. Like Robertson's, this article can be considered an array cooking lesson. Familiarity with basic array functionality is, therefore, assumed.
3. I found myself needing square tables only to faithfully reproduce work from another author.
4. For a detailed discussion of what constitutes a 'good' table, please refer to the booktab package documentation.

quick example that struts the first row:

X	Y	Z
Z	X	Y

```
\begin{tabular}{l|l|l}
  {\large\strut} X & Y & Z \\
                 Z & X & Y
\end{tabular}
```

Aside from struts, commands such as `\arraystretch` and `\extrarowheight` are available. They are not suitable for columns set in paragraph mode, because the row height is scaled with respect to individual row contents when using either of these methods. When uniformity is desired, the scaling must not vary in that manner.

Given struts, all we would have to do is figure out exactly how high our rows should be and adjust the struts' height accordingly. Unfortunately, this approach does not work as expected because the strut influences the other contents of a cell and vice versa. This is because struts are designed to work on a *single* line. This would be fine if we had single-line cells since any adjustments would be the same for each cell. For multiline cells, adjustments would vary depending on the number of lines present.

3 Minipages?

One solution that comes to mind for multi-line cells is to add another column containing only struts.[5] However, I try to avoid mixing typesetting logic with my tabular data. A better approach is to subdivide the contents of the cell into two minipages. Using the array package, we can keep this logic well away from our tabular data.

Flawed solution

```
\newcolumntype{s}[1]{%
  >{\begin{minipage}{0pt}%
    \rule{0pt}{#1 + 2\tabcolsep}%
    \end{minipage}%
    \begin{minipage}[c]{#1}%
    \centering\arraybackslash }
  m{#1}
  <{\end{minipage}}
```

5. This interferes with automatic width calculation by e.g. tabularx.

```
\begin{tabular}{*{3}{|s{4em}}|}
  ... table data goes here ...
\end{tabular}
```

You'll note that instead of a `\strut` I'm using a zero-width `\rule`, which takes the desired height as its second argument.

The approach using minipages produces output that's exactly what we were looking for. But it has the downside of being a bit clumsy; it makes the `m{...}` specifier all but useless, and let's face it: wrapping the cell contents isn't playing nicely.

Regular users of `array` will remember that the `@{...}` column specifier replaces the inter-column spacing. We can use this to our advantage to produce output exactly like the minipage example. We use a `\parbox` to replace the spacing we normally get.

Preferred solution

```
\newcolumntype{z}[1]{
  @{{\centering\parbox[c]{\tabcolsep}{\rule{0pt}{#1 + 2\tabcolsep}}}}
  >{\centering\arraybackslash}
  m{#1}}

\begin{tabular}{*{3}{|z{4em}}|}
  ... table data goes here ...
\end{tabular}
```

4 Using tabularx

Some users may prefer to specify the width of the whole table rather than the column widths. Users of tabularx can still create square, multi-line cells even while individual column widths are calculated for them.

We are very fortunate that the user is exposed to the calculated width so it can be used in LaTeX code directly. In order to control vertical alignment, tabularx allows the user to renew the `\tabularxcolumn` command, which takes as its argument the width tabularx calculated. By default, it is defined as `\newcommand{\tabularxcolumn[1]{p{#1}}`.

We can take advantage of this command to make our cells exactly tall enough. Using the column type *z* defined above we would do something like:

Partial example

```
\renewcommand{\tabularxcolumn}[1]{z{#1}}

\begin{tabularx}{.5\textwidth}{|X|X|X|}
   ... table contents go here ...
\end{tabularx}
```

A complete listing of this code can be found in section 7.

5 Caveat emptor

Certainly, the solutions offered in this article are not perfect. Most importantly, the user must make sure that enough space has been allocated to contain the contents of the square cells. LaTeX will still produce uneven cells if the contents will not fit in a square area corresponding to the specified tabular column width or tabularx table width.

Additionally, the approach taken is intended for columns typeset in paragraph mode (i.e., using the `p{...}`, `m{...}`, or `b{...}` column specifiers) from the `array` package.

Users requiring more advanced formatting options are well advised to consider graphics packages such as PGF.

6 Summary

Multiline square cells typeset in paragraph mode require special handling. Minipages can be utilised to subdivide cells so that cell contents won't foil the strutting. A cleaner approach is to use the `@{...}` **array specifier** to prepend the strut to every cell. This solution works with both array-enabled `tabular` and the tabularx package. A full example containing both uses is provided.

7 A minimal example

```
\documentclass[a4paper, 12pt]{article}
\usepackage{array}
\usepackage{tabularx}
\usepackage{calc}

\newcolumntype{z}[1] {
  @{{\centering \parbox[c]{\tabcolsep}{\rule{0pt}{#1 + 2\tabcolsep}}}}
  >{\centering\arraybackslash}
  m{#1} }

\renewcommand{\tabularxcolumn}[1]{z{#1}}

\newcommand{\tabledata}{
\hline Lorem ipsum dolor sit & amet, consectetur \ldots %
& elit, sed do eiusmod \ldots \\ \hline
ut labore et dolore & magna aliqua. & Ut enim  \\ \hline
ad minim veniam quis & nostrud exercitation & ullamco laboris \\ \hline
}

\begin{document}

\begin{tabularx}{.5\textwidth}{|X|X|X|}
  \tabledata
\end{tabularx}
\qquad
\begin{tabular}{*{3}{|z{4em}}|}
  \tabledata
\end{tabular}

\end{document}
```

which produces:

Lorem ipsum dolor sit	amet, consectetur ...	elit, sed do eiusmod ...
ut labore et dolore	magna aliqua.	Ut enim
ad minim veniam quis	nostrud exercitation	ullamco laboris

Lorem ipsum dolor sit	amet, consectetur ...	elit, sed do eiusmod ...
ut labore et dolore	magna aliqua.	Ut enim
ad minim veniam quis	nostrud exercitation	ullamco laboris