The PracTEX Journal
TPJ 2005 No 02, 2005-04-15
Rev. 2005-03-25

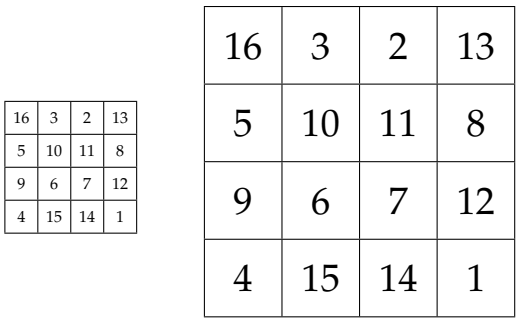
Square cells: an array cooking lesson

Will Robertson

1 Square cells in tabular

Late last year, there was a question on the Mac OS X TEX mailing list asking if it were possible to create a table with square cells. My solution made extensive use of the array package (as well as some basic LATEX programming concepts), so it seemed appropriate to make something of a tutorial out of it. So here we are!

In typical style, I shall begin with the final result, followed by a detailed description of the implementation. Admire the squarecells environment:



16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

```
\newcommand\example{%
  \begin{squarecells}{4}
    16 & 3  & 2  & 13 \nl
    5  & 10 & 11 & 8  \nl
    9  & 6  & 7  & 12 \nl
    4  & 15 & 14 & 1  \nl
  \end{squarecells}%
}\tiny \example \hspace{20pt}
 \large\example
```

You can see that it scales with font size, the numbers are centered both horizontally and vertically, and the lines are generated automatically with the use of \nl to end each row. The only input to the environment is the number of columns in the array. Very simple!

Before we begin, I might encourage the reader to typeset the example document distributed with this article; if it fails to compile, you may wish to update (at least) your copy of array.sty, which had its most recent revision in Dec. 2003.

2 Creating square cells from tabular

Out of the box, LaTeX's tabular facilities are useful but not particularly flexible. The array package changes this by providing tools to allow tables that can be formatted in very many ways. (Maths arrays are also typeset using the same features as tabulars, hence the name.) The tabular environment looks something like this:[1]

```
\begin{tabular}{column preamble}
    tabular material
\end{tabular}
```

The *column preamble* defines the number of columns, the behaviour of each column, and the formatting of each column. Within the *tabular material*, columns are separated by `&` and rows are separated by `\\`.

There are two types of column specifiers we may use to define the behaviour of the cells: single line stretch-to-fit; or fixed-width wrapping paragraph. The differences in the various paragraph column specifiers do not interest us for our application – we may use simply `p{width}`. This defines a column of specific width in which text will wrap if there is not enough space. For *our* use in creating square cells, we also hope that the text won't wrap; it is the fixed width that interests us!

2.1 Setting the column widths

We start off by defining how large each cell of the tabular should be; this dimension will be stored in `\celldim`[2]. We shall define it as 2 em, which gives us enough space for a character or two (you might like to increase it if you need more space). The em unit scales with font size so we don't have to worry about changing the definition in case the environment is used with a different font size.

To create vertical lines in a tabular, `|` is used in the column preamble; `\hline` in the tabular itself creates a horizontal line. We can now place the first piece of our puzzle:

[1] Please see one of the various LaTeX manuals for a more thorough reference, including:
- *The LaTeX Companion* (2nd Ed., chapter 5), Frank Mittelbach and Michel Goossens *et al.*;
- *A Guide to LaTeX* (4th Ed., chapter 6), Helmut Kopka and Patrick Daly; and
- *LaTeX: A Document Preparation System* (2nd Ed., appendix C.10.2), Leslie Lamport.

[2] Lengths must be defined (*e.g.*, `\newlength\celldim`) before they may be used; this has been omitted from the examples in order to save space.

1	2	3
4	5	6
7	8	9


```
\setlength\celldim{2em}
\begin{tabular}
    {|p{\celldim}|p{\celldim}|p{\celldim}|}
\hline  1 & 2 & 3   \\   \hline
        4 & 5 & 6   \\   \hline
        7 & 8 & 9   \\   \hline
\end{tabular}\vspace{1em}

\rule[-3pt]{1pt}{7pt}\rule{2em}{1pt}%
\rule[-3pt]{1pt}{7pt}\\    2\,em
```

You'll notice, however, that the cells are in fact *wider* than 2 em since tabular inserts some space before and after each column. This can be fixed using the `@{...}` column specifier, which inserts its argument instead of the normal intercolumn space in the position at which it is located. This can be used to vary the intercolumn space or to insert material in between columns. When its argument is empty (`@{}`), we remove the intercolumn space and insert nothing, resulting in the desired behaviour: each column is exactly the width we specify with no surrounding space.[3]

We will use this next, but before we do, we can save space and sanity when typing the column preamble by defining our own column specifiers:

`\newcolumntype{`*column specifier*`}{`*column definition*`}`

Rather than having to repeatedly include every *column definition* for every single column we wish to have in the table, `\newcolumntype` allows us to define all of the formatting in advance and use our new *column specifier* instead.

So, by specifying the width of each column and suppressing the intercolumn space, each cell is now exactly `\celldim` wide:

1	2	3
4	5	6
7	8	9

```
\setlength\celldim{2em}
\newcolumntype{S}{@{} p{\celldim} @{}}
\begin{tabular}{|S|S|S|}
\hline  1 & 2 & 3   \\   \hline
        4 & 5 & 6   \\   \hline
        7 & 8 & 9   \\   \hline
\end{tabular}
```

[3] The amount of space inserted by default is controlled by the `\tabsepcol` length, so another method to fix the problem could be to locally set it to zero in our table.

2.2 Centering the text horizontally

Next we add horizontal centering by applying a `\centering` command to every cell. The array package provides the `>{`*stuff to insert*`}` column specifier which adds the *stuff to insert* to the beginning of each cell in the next column. In our case, we simply want a `\centering` command, but because `\centering` redefines `\\` to something else, we also need the `\arraybackslash` command to define it back to its original definition. So, now we have centered cells:

1	2	3
4	5	6
7	8	9

```
\setlength\celldim{2em}
\newcolumntype{S}
  {@{}
   >{\centering\arraybackslash}
   p{\celldim} @{}}
\begin{tabular}{|S|S|S|} \hline
        1 & 2 & 3   \\   \hline
        4 & 5 & 6   \\   \hline
        7 & 8 & 9   \\   \hline
\end{tabular}
```

2.3 Setting the row heights & vertical centering

All that's left to create the squares is to set the height of each cell equal to the width. This can be achieved by inserting a strut (a zero-width rule with vertical height) of a certain height and a certain depth into each row.

To determine the amount of extra space we need to insert into the row, we need to measure the height of the text that will already be there. We perform this with the `\settoheight` command, storing the resulting length in `\fontheight` (not forgetting that lengths require initialising).

Since we want the text in the cells to be vertically centered, we need to add an equal space above and below the space taken up by the characters. This space is half the difference of `\fontheight` and `\celldim` to create squares. For the calculation of lengths, the calc package here makes life a lot easier for us, allowing direct manipulation of the various length commands.

The strut is created with `\rule`, which takes first an optional argument to shift it from the baseline (positive is raise, negative is lower), then two mandatory arguments for its width (which will be zero to make it invisible) and height.

We use the `<{...}` specifier to add our strut to the end of each cell in exactly the same way as `>{...}` was used.[4] Finally, we have our square cells:

1	2	3
4	5	6
7	8	9

```
\setlength\celldim{2em}
\settoheight\fontheight{A}
\setlength\extraheight{\celldim - \fontheight}
\newcolumntype{S}
  { @{}
    >{\centering\arraybackslash}
    p{\celldim}
    <{\rule[-0.5\extraheight]{0pt}%
           {\fontheight + \extraheight}}
    @{} }
\begin{tabular}{|S|S|S|} \hline
        1 & 2 & 3   \\   \hline
        4 & 5 & 6   \\   \hline
        7 & 8 & 9   \\   \hline
\end{tabular}
```

3 Creating the environment

Well, we can now typeset tables with square cells. But we certainly don't want to have to copy/paste that big chunk of LaTeX code every time we need to use one! So, the next thing to do is to define a new environment, which we shall call squarecells, for using these tables with a minimum of fuss.

3.1 The beginning of the end

Essentially, we will just take all of the previously written code and wrap it up into an environment definition. The `\newcolumntype` definition of the `S` column specifier does not need to be included, however, since it will never change. The other commands *are* included, since they work with lengths that are specified in ems, which will vary if we change fonts or font sizes.

[4] There's no reason why we couldn't have put the strut in our `>{...}` specifier, but I preferred splitting the column specifiers up by functionality.

To create the environment, we use the `\newenvironment` command:

`\newenvironment{`*env. name*`}[`*no. of args*`]{`*before code*`}{`*after code*`}`

So, two features are put into the environment to allow it to work nicely. Rather than making the user bother with typing in the tabular preamble `|S|S|S|S...` for as many columns required, it would be better to have an argument to the environment to declare the number of columns in the table. Luckily, this is easy with the column specifier `*{`*N*`}{`*repeat definition*`}` that simply replaces itself with *N* copies of *repeat definition*.

Second, it's annoying having to manually place all of those horizontal lines. The quick-and-easy solution is to define a new command `\nl` (short for *n*ew *l*ine) which incorporates both an 'end-of-row' command and a horizontal line. This then requires that the first `\hline` before the table itself is placed in the environment definition. This is it:

```
% \newcolumntype{S}{...} as defined earlier
\newenvironment{squarecells}[1]
  {\setlength\celldim{2em}%
   \settoheight\fontheight{A}%
   \setlength\extraheight{\celldim - \fontheight}%
   \begin{tabular}{ |*{#1}{S|} } \hline }
% squarecells tabular goes here
  {\end{tabular}}
\newcommand\nl{\\\hline}
```

Now we have technically everything necessary to typeset the example given on the first page. That wasn't so bad, huh? Well, there are actually a few loose ends that could be tied up if we wished, so the less adventurous may not wish to continue. . . .

3.2 The more complicated finish

Back when we defined our own column specifier, we added in some code which inserts a rule into every single cell to achieve the desired row height to create square cells. Well, it's only actually necessary to add a strut to a *single* cell per row in order to get the desired effect, since a `tabular` row stretches to fit in the tallest cell. In the interest of efficiency, let's make our tabular environment insert the strut only once per row.

So, we'll create a new column type, `Z`, which will be used for all columns after the first (which will still be `S`), and which doesn't insert the strut. Our tabular preamble will then be `|S|*{n-1}{Z|}`, where `n` is the number of columns specified in the argument of `squarecells`. To implement this, we need a new counter, called `sqcolumns`,[5] to store `n-1`.

```
% \newcolumntype{S}{...} as defined earlier
\newcolumntype{Z}{ @{} >{\centering} p{\celldim} @{} }
\newenvironment{squarecells}[1]
  {\setlength\celldim{2em}%
   \settoheight\fontheight{A}%
   \setlength\extraheight{\celldim - \fontheight}%
   \setcounter{sqcolumns}{#1 - 1}%
   \begin{tabular}
     {|S|*{ \value{sqcolumns} }{Z|}}
   \hline}
% square cell tabular goes here
  {\end{tabular}}
\newcommand\nl{\tabularnewline\hline}
```

Did you spot that there was also one other small change that wasn't coloured in? Rather than redefining `\\` in every single cell with `\arraybackslash` in the `>{}` column specifier (this is wasteful since `\\` isn't used until the end of the row), we can simply use the longhand 'new row' command `\tabularnewline` in the definition of `\nl`. (This change would also affect the definition of `S`.)

Now, once again, we're ready to go back to the first page and see the fruits of all this labour. So, perhaps being able to typeset tables with square cells won't be that useful particularly often to some people, but the process of explaining their implemention should prove to have been more helpful.

On the next page, we conclude with a minimal example demonstrating the techniques performed in this article. Looking through the code without referring back to the notes, it should now, hopefully, all make some kind of sense. Any feedback the reader may wish to contribute is greatly encouraged, particularly for alternate methods of solving the problem. I hope you've had fun!

[5] This is defined in the preamble with `\newcounter{sqcolumns}`.

4 A minimal squarecells-example.tex

```
\documentclass{article}
\usepackage{array}[2003/12/17]
\usepackage{calc}

\newlength\celldim \newlength\fontheight \newlength\extraheight
\newcounter{sqcolumns}

\newcolumntype{S}{ @{}
  >{\centering \rule[-0.5\extraheight]{0pt}{\fontheight + \extraheight}}
  p{\celldim} @{} }

\newcolumntype{Z}{ @{} >{\centering} p{\celldim} @{} }

\newenvironment{squarecells}[1]
  {\setlength\celldim{2em}%
   \settoheight\fontheight{A}%
   \setlength\extraheight{\celldim - \fontheight}%
   \setcounter{sqcolumns}{#1 - 1}%
   \begin{tabular}{|S|*{\value{sqcolumns}}{Z|}}\hline}
% squarecells tabular goes here
  {\end{tabular}}

\newcommand\nl{\tabularnewline\hline}

\begin{document}
  \Huge
  \begin{squarecells}{4}
    16 & 3  & 2  & 13 \nl
    5  & 10 & 11 & 8  \nl
    9  & 6  & 7  & 12 \nl
    4  & 15 & 14 & 1  \nl
  \end{squarecells}
\end{document}
```