Building Complex Tables in LaTeX

Benjamin Nutter

December 4, 2015

Contents

Introduction

When I first started writing `pixiedust`, I chose to approach it by first writing the HTML tables, even though what I really wanted for my professional documents was the PDF output. I figured that if I could do a proof of concept in HTML, then I would probably be able to figure out how to get similar PDF output. Unfortunately, getting the matching PDF output has turned out to be more difficult than I expected.

After making a couple of attempts at building the complex LaTeXtables, I decided it would be in my best interest to document how I came about the solutions I implemented to build these tables. Doing so will be important to help remind myself about the decisions I made along the way that were necessary to make everything work together.

Summary of Chosen Behaviors

Horizontal Alignment	When no column widths, row heights, or vertical alignments are requested in a column, horizontal aligments will be managed with the `l`, `r`, and `c` column tags. If any column width, height, or vertical alignment is provided in a column, the horizontal alignment will be managed via `raggedright`, `centering`, and `raggedleft` tags in the `parbox`

Column Width	Column widths will be determined by the largest cell width defined within a column. All widths will be converted to `pt` before determining which is the largest. In the event that no column width is specified but one is required (due to a condition that requires the use of a `parbox`), the column width will be estimated based on the maximum number of characters in the column cells. The estimate will be 10 `pt` per character

The Most Basic of Tables

Simple Tables

The most simple tables in LaTeXare quite simple to construct. The primary components are

- Beginning table tag (such as `\begin{tabular}` or `begin{longtable}`
- Column alignments (`l`, `c`, `r`, `p`, `m`, `b`; use of `p`, `m`, and `b` requires the `array` package)
- Row contents
- Ending table tag (such as `\end{tabular}` or `\end{longtable}`

In each row, the contents of the columns are divided by the `&` symbol, and the end of the row is marked by the `\\` symbol.

The column alignments have the following behaviors:

- `l`: horizontally aligns the cell text to the left
- `c`: horizontally aligns the cell text to the center
- `r`: horizontally aligns the cell text to the right
- `p`: vertically aligns the first line of text to the center of the row, with left horizontal alignment (accepts (requires?) a width parameter)
- `m`: vertically aligns the middle of the text to the center of the row, with left horizontal alignment (accepts (requires?) a width parameter)
- `b`: vertically aligns the bottom line of text to the center of the row, with left horizontal alignment. (accepts (requires?) a width parameter)

More on `p`, `m`, and `b`

Another aspect of the `p`, `m`, and `b` column types is that they allow for multiple lines of text in the cell, whereas `l`, `c`, and `r` will spread the width of the cell to the width of the text. In order to wrap longer text into multiple lines within a cell, one must use a width parameter with `p`, `m`, or `b`.

Consider the following table:

```
\begin{tabular}{|l|c|r|p{1cm}|m{1cm}|b{1cm}|} \hline
Column 1 & Column 2 & Column 3 & Column 4 & Column 5 & Column 6 \\ \hline
One      & Two      & Three    & Four     & Five     & Six   \\ \hline
\end{tabular}
```

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
One	Two	Three	Four	Five	Six

Vertical Alignment

Mechanics

This previous table example doesn't exactly let us see the effect of `p`, `m`, and `b`, so let's define another table that shows the behavior a little better. This table is copied from a Stack Exchange question [1].

```
\begin{tabular}{|p{1.5in}|m{1.5in}|b{1.5in}|}
\hline
\centering header p &
\centering header m &
\centering header b \tabularnewline
\hline
text which is considerably longer than the width of the column  &
text which is considerably longer than the width of the column  &
text which is considerably longer than the width of the column
\tabularnewline
\hline
\end{tabular}
```

header p	header m	header b
text which is considerably longer than the width of the column	text which is considerably longer than the width of the column	text which is considerably longer than the width of the column

Again, notice that the top line of the left column is aligned to the center of the row; the middle of the center column is aligned to the center of the row; and the bottom line of the cell is aligned to the center of the row. All of the text is horizontally aligned to the left.

Another enormously important observation for these table tags is that they cannot guarantee that the contents of a cell are vertically aligned to the top or bottom (that is, that the first line of text is always at the top of the cell, or the last line of text is always at the bottom of the cell). Notice what happens if we try to have two columns centered at the bottom but where one column has longer text than the other.

```
\begin{tabular}{|m{1.5in}|b{1.5in}|b{1.5in}|}
\hline
\centering header m &
\centering header b &
\centering header b \tabularnewline
\hline
text which is considerably longer than the width of the column  &
text which is considerably longer than the width of the column and repeat
text which is considerably longer than the width of the column  &
text which is considerably longer than the width of the column
\tabularnewline
\hline
\end{tabular}
```

header m	header b	header b
text which is considerably longer than the width of the column	text which is considerably longer than the width of the column and repeat text which is considerably longer than the width of the column	text which is considerably longer than the width of the column

It is worth noting, however, that this problem only tends to manifest when you have differing vertical alignments within a row. When all of the vertical alignments in a row are the same, you may expect to get the desired vertical alignment. Consider the following examples:

```
\begin{tabular}{|b{1.5in}|b{1.5in}|}
\hline
\centering header b &
\centering header b \tabularnewline
\hline
text which is considerably longer than the width of the column and repeat
text which is considerably longer than the width of the column  &
text which is considerably longer than the width of the column
\tabularnewline
\hline
\end{tabular}
```

header b	header b
text which is considerably longer than the width of the column and repeat text which is considerably longer than the width of the column	text which is considerably longer than the width of the column

```
\begin{tabular}{|p{1.5in}|p{1.5in}|}
\hline
\centering header p &
\centering header p \tabularnewline
\hline
text which is considerably longer than the width of the column and repeat
text which is considerably longer than the width of the column  &
text which is considerably longer than the width of the column
\tabularnewline
\hline
\end{tabular}
```

header p	header p
text which is considerably longer than the width of the column and repeat text which is considerably longer than the width of the column	text which is considerably longer than the width of the column

Altering Alignment From One Row to the Next

While we can get the desired vertical alignment when all of the columns show the same vertical alignment, the `p`, `m`, and `b` column tags define the entire column. If we wish to change the vertical alignment from one row to the next, we will have to use the `\parbox` command. The `t`, `c`, and `b` alignments for `\parbox` are analagous to `p`, `m`, and `b`, respectively.

```
\begin{tabular}{|l|p{1.5in}|p{1.5in}|}
\hline
\centering header p &
\centering header p \tabularnewline
\hline
top aligned &
text which is considerably longer than the width of the column and repeat
text which is considerably longer than the width of the column  &
text which is considerably longer than the width of the column
\tabularnewline
\hline
center aligned &
\parbox[c]{1.5in}{text which is considerably longer than the width of the column and repeat
text which is considerably longer than the width of the column}  &
\parbox[c]{1.5in}{text which is considerably longer than the width of the column}
\tabularnewline
\hline
bottom aligned &
\parbox[b]{1.5in}{text which is considerably longer than the width of the column and repeat
text which is considerably longer than the width of the column}  &
\parbox[b]{1.5in}{text which is considerably longer than the width of the column}
\tabularnewline
\hline
\end{tabular}
```

header p	header p	
top aligned	text which is considerably longer than the width of the column and repeat text which is considerably longer than the width of the column	text which is considerably longer than the width of the column
center aligned	text which is considerably longer than the width of the column and repeat text which is considerably longer than the width of the column	text which is considerably longer than the width of the column
bottom aligned	text which is considerably longer than the width of the column and repeat text which is considerably longer than the width of the column	text which is considerably longer than the width of the column

One of the side effects of using `\parbox` however is that it requires a width argument. In order to get the width to match with the rest of the column, `pixiedust` has to find a common column width for all cells in that column. This is discussed in the next section.

pixiedust and Choosing the Column Width

The pixiedust philosophy of building tables is that the user should be able to modify any one cell in a table without minimal side effects to the rest of the table. The first instance in which difficulties arise with this approach is in column widths. If the user chooses to define a cell's width, one of the p, m, or b tags must be used. If the user chooses to define the cell widths of two cells in one column, there is a potential for a conflict. HTML resolves this conflict automatically, and will use the largest column width when two width occur together. The pixiedust approach to LaTeX, however requires that pixiedust know the width of a column prior to building the cells (this isn't always necessary in pure LaTeX, but will become essential when trying to combine both \multirow and \multicolumn).

In order to behave most similarly to the HTML output, pixiedust will scan the widths of every cell in a column and look for the largest width. The entire column will be defined with that width.

The other wrinkle in this process is that widths can be specified in multiple units. The acceptable LaTeXunits are pt, cm, in, and % [1] The challenge, then, is to convert all of the varying units into a single common unit for comparison. pixiedust will convert everything to pt. When width_units = %, the width of the table will be taken from the dust objects table_width element (which defaults to six inches). It is assumed that there are 72.27 pt in an in, and 28.45 pt in a cm [2].

After converting all of the cell widths in a column into a common unit, pixiedust simply chooses the largest one and sets the column width accordingly using the p tag. (Vertical alignments are handled in pixiedust using parbox)

References

1. http://tex.stackexchange.com/questions/35293/p-m-and-b-columns-in-tables (Cell widths were modified from the original example)
2. http://tex.stackexchange.com/questions/8260/what-are-the-various-units-ex-em-in-pt-bp-dd-pc-expressed-in-mm

[1] LaTeXwill recognize other units, but ‘pixiedust‘ chooses not to recognize them. Also, ‘pixiedust‘ recognizes ‘px‘, but in LaTeXoutput, will convert it to ‘pt‘.