Authoritative LaTeX

Power LaTeX for Power Authors

Deyan Ginev

Jacobs University Bremen

1 Quick Intro

On May 1st, 2015, Authorea deployed a new backend for its LaTeX input language, teaming up with the ambitious LaTeXML project, which strives to offer a full reimplementation of TeX with targeted generation of web-first manuscripts, supporting HTML5 and ePub.

To enable LaTeXML, select the following option in your article page:

Settings (topbar Gear icon) → Advanced settings → More Preferences → LaTeX Support → LaTeXML

In this article you can find an overview of some of our new high impact authoring features.

2 Code Listings

We will be illustrating all of our examples with code listings in this tutorial, so let's first show how to set them up.

An example setup which you can add to your `header.tex` file is:

```
\usepackage{listings}
\lstset{ %
  backgroundcolor=\color{white},   % choose the background color
  basicstyle=\footnotesize,        % size of fonts used for the code
  breaklines=true,                 % automatic line breaking only at whitespace
  captionpos=b,                    % sets the caption-position to bottom
  commentstyle=\color{OliveGreen},    % comment style
  keywordstyle=\color{BlueViolet},       % keyword style
  stringstyle=\color{black},     % string literal style
  language=[AlLaTeX]TeX,             % Set your language (you can change the language
      for each code-block optionally)
  frame=lrtb, %
  xleftmargin=\fboxsep, %
  xrightmargin=-\fboxsep, %
  moretexcs={lstset,color,colorlet, cellcolor, newcolumntype, columncolor, rowcolor,
      multirow, xspace, LaTeX, TeX},
}
```

Useful to keep in mind: definitions in your `header.tex` file are global to your article, while macros introduced in a paragraph are only local to that paragraph. That applies even to global commands such as `\gdef`, as Authorea processes each paragraph independently of its surroundings.

3 Basic Macros

As mentioned above, only definitions placed in `header.tex` are global for your entire article. You can use all flavors of TEX's definitions (`\def`, `\edef`, `\xdef`, `\gdef`, `\newcommand`, `\rewnewcommand`, ...) to your liking.

If you want to discuss a notable **term**, such as LaTeXML, you may want to define a handy macro which:

1. Links to the term's Wikipedia page
2. Is available in your entire document (define it in `header.tex`)
3. Flexibly figures out if it needs a trailing space or not[1]
4. Is easier to write than the term's full name

Here is one way you can achieve that, with the entities used in this tutorial:

```
% All \usepackage requirements MUST be placed in header.tex:
\usepackage{xspace}
% as well as any definitions and settings inteded for your entire article:
\def\authorea{\href{http://en.wikipedia.org/wiki/Authorea}{Authorea}\xspace}
\def\latexml{\href{https://en.wikipedia.org/wiki/LaTeXML}{LaTeXML}\xspace}
\def\latex{\href{https://en.wikipedia.org/wiki/LaTeX}{\LaTeX}\xspace}
% ... later, use those definitions in an Authorea paragraph as:
\begin{center}
\authorea + \latexml + \latex = {\color{BrickRed}{}}
\end{center}
```

Authorea + LaTeXML + LaTeX =

[1] one of TEX's original weaknesses with its macros

4 Text Styles

4.1 Fonts

`\normalfont` (default)	Authorea
`\rmfamily`	Authorea
`\sffamily`	Authorea
`\ttfamily`	Authorea

4.2 Sizes

`\Huge`	Authorea
`\huge`	Authorea
`\LARGE`	Authorea
`\Large`	Authorea
`\large`	Authorea
`\normalsize` (default)	Authorea
`\small`	Authorea
`\footnotesize`	Authorea
`\scriptsize`	Authorea
`\tiny`	Authorea

4.3 Text Color

If you want the extended color set, you will need to add `\usepackage[dvipsnames,tables]{xcolor}` to your `header.tex` file.

The 68 standard colors known to dvips are (in alphabetical order):

Apricot	Aquamarine	Bittersweet	Black
Blue	BlueGreen	BlueViolet	BrickRed
Brown	BurntOrange	CadetBlue	CarnationPink
Cerulean	CornflowerBlue	Cyan	Dandelion
DarkOrchid	Emerald	ForestGreen	Fuchsia
Goldenrod	Gray	Green	GreenYellow
JungleGreen	Lavender	LimeGreen	Magenta
Mahogany	Maroon	Melon	MidnightBlue
Mulberry	NavyBlue	OliveGreen	Orange
OrangeRed	Orchid	Peach	Periwinkle
PineGreen	Plum	ProcessBlue	Purple
RawSienna	Red	RedOrange	RedViolet
Rhodamine	RoyalBlue	RoyalPurple	RubineRed
Salmon	SeaGreen	Sepia	SkyBlue
SpringGreen	Tan	TealBlue	Thistle
Turquoise	Violet	VioletRed	
WildStrawberry	Yellow	YellowGreen	YellowOrange

Use them via `{\color{ColorName} your text}` .

5 Definitions, Theorems, Lemmas and more

AMS-flavored LaTeX introduces a flexible language for declaring custom types of environments. It has long been the weapon of choice for declaring mathematical statements, for example:

```
% Declare in header.tex:
\newtheorem{theorem}{Theorem}
\newtheorem{proof}{Proof}
% ...
% And later use:
\begin{theorem}[Pythagoras]
  Suppose $a\leq b\leq c$ are the side-lengths of a right triangle.\\
  Then $a^2+b^2=c^2$.
\end{theorem}
\begin{proof}
  Proof is left as an exercise to the reader.
\end{proof}
```

Theorem 1 (Pythagoras). *Suppose $a \leq b \leq c$ are the side-lengths of a right triangle. Then $a^2 + b^2 = c^2$.*

Proof. Proof is left as an exercise to the reader.

6 Tables

6.1 Merging rows and columns

- To merge columns: `\multicolumn{cell count}{layout}{content}`
- To merge rows: `\multirow{cell count}{layout}{content}`

Example:

```
\begin{tabular}{|l|l|l|}
\hline
\multicolumn{3}{|c|}{Table cheat sheet} \\
\hline
Horizontal line & \verb|\hline| & underlines current table
    row \\ \hline
\multirow{4}{*}{Column types} & \verb|l| & left \\
 & \verb.c. & center\\
 & \verb.r. & right\\
 & \verb,p{'width'}, & custom width, aligned top \\
 & \verb,m{'width'}, & custom width, aligned middle \\
 & \verb,b{'width'}, & custom width, aligned bottom \\ \
     hline
\multirow{3}{*}{Borders} & none & default \\
 & \verb.|. & single, as in \verb.l|. \\
 & \verb.||. & double, as in \verb.l||c. \\ \hline
Alignment & \& & separates columns \\ \hline
\multirow{2}{*}{End of line} & \verb|\\| & new table row \\
 & \verb|newline| & new line within cell \\
\hline
\end{tabular}
```

Table cheat sheet		
Horizontal line	`\hline`	underlines current table row
Column types	`l`	left
	`c`	center
	`r`	right
	`p{'width'}`	custom width, aligned top
	`m{'width'}`	custom width, aligned middle[a]
	`b{'width'}`	custom width, aligned bottom[b]
Borders	none	default
	`|`	single, as in `l|`
	`||`	double, as in `l||c`
Alignment	&	separates columns
End of line	`\\`	new table row
	`newline`	new line within cell

[a] requires array.sty
[b] requires array.sty

6.2 Advanced Table Borders

Sometimes it is helpful to be able to provide partial border specifications, for either rows or columns.

- `\cline{i-j}` provides a partial bottom border between columns `i` and `j` of the current row.
- A combined use of `\multicolumn` and `\multirow` can build a partial vertical border.

Here is one illustrating example:

```
\begin{tabular}{cc|c|c|c|c|l}
\cline{3-6}
& & \multicolumn{4}{ c| }{Primes} \\ \cline{3-6}
& & 2 & 3 & 5 & 7 \\ \cline{1-6}
\multicolumn{1}{ |c  }{\multirow{2}{*}{Powers} } &
\multicolumn{1}{ |c| }{504} & 3 & 2 & 0 & 1 &     \\ \cline
    {2-6}
\multicolumn{1}{ |c  }{}                          &
\multicolumn{1}{ |c| }{540} & 2 & 3 & 1 & 0 &     \\ \cline
    {1-6}
\multicolumn{1}{ |c  }{\multirow{2}{*}{Powers} } &
\multicolumn{1}{ |c| }{gcd} & 2 & 2 & 0 & 0 & min \\ \cline
    {2-6}
\multicolumn{1}{ |c  }{}                          &
\multicolumn{1}{ |c| }{lcm} & 3 & 3 & 1 & 1 & max \\ \cline
    {1-6}
\end{tabular}
```

		Primes			
		2	3	5	7
Powers	504	3	2	0	1
	540	2	3	1	0
Powers	gcd	2	2	0	0	min
	lcm	3	3	1	1	max

Original source at wikibooks.org

6.3 Advanced Table Colors

Row Colors In Section 4.3 we showed the extended set of colors available through the `xcolor.sty` package. For tables, it is also useful to have control over the background color of individual rows. One can set row colors via the aptly named `\rowcolor` macro, and set alternating colors for rows using the `\rowcolors` macro, starting at a particular row offset. Here is an example:

```
\colorlet{MyGreen}{green!80!yellow!50}
\colorlet{MyLighterGreen}{green!70!yellow!40}
\rowcolors{3}{MyGreen}{MyLighterGreen}
\begin{tabular}{|c|}
\hline
An Example of Row Colors \\
\hline
Heading can be White \\
\hline
Green \\
Lighter Green \\
Green \\
Lighter Green \\
\hline
\end{tabular}
```

An Example of Row Colors
Heading can be White
Green
Lighter Green
Green
Lighter Green

Backgrounds for Individual Cells Setting the background color of a column can be achieved indirectly via the `\newcolumntype` macro, which defines a new column type. That new type could be based on an existing column specification (e.g. the "centered" `c`), with a modified background color via the `\columncolor` macro. In case a row or column background color needs to be overridden, the cell-specific `\cellcolor` macro comes to the rescue. Here is an illustrating example that puts all of these techniques together:

```
% Define some custom colors
\colorlet{FreshGray}{gray!20!white}
\colorlet{FreshGreen}{green!20!white}
\colorlet{FreshYellow}{yellow!20!white}
\colorlet{FreshRed}{red!20!white}
% Define shorthand macros for coloring individual cells
\def\okcell{\cellcolor{FreshGreen}}
\def\avgcell{\cellcolor{FreshYellow}}
\def\badcell{\cellcolor{FreshRed}}
\def\nocell{\cellcolor{FreshGray}}

% A new column type, adding a FreshGray background to the centered type
\newcolumntype{g}{>{\columncolor{FreshGray}}c}
\begin{tabular}{ |g|c|c|c|c|  }
\hline
\multicolumn{5}{|c|}{Comparison of Sorting Algorithms} \\
\hline \rowcolor{FreshGreen}
  \multirow{2}{*}{\okcell Name} & \multicolumn{3}{|c|}{Performance} & \multirow
      {2}{*}{\okcell Memory} \\ \cline{2-4}
   & Best & Average & Worst &  \\
\hline
Quicksort & \okcell $n \log(n)$ & \okcell $n \log(n)$ & \badcell $n^2$ & \avgcell $\
    log n$ \\
Merge Sort & \okcell $n \log(n)$ & \okcell $n \log(n)$ & \okcell $n \log(n)$ & \
    badcell $n$ worst case\\
In-place merge Sort & \nocell — & \nocell — & \avgcell $n \log^2(n)$ & \okcell 1 \\
\hline
\end{tabular}
```

Comparison of Sorting Algorithms				
Name	Performance			Memory
	Best	Average	Worst	
Quicksort	$n \log(n)$	$n \log(n)$	n^2	$\log n$
Merge Sort	$n \log(n)$	$n \log(n)$	$n \log(n)$	n worst case
In-place merge Sort	–	–	$n \log^2(n)$	1

Source for the tabular data: http://en.wikipedia.org/wiki/Sorting_algorithm

7 Layout Directives

7.1 General Layout

```
{\centering You can display text:}
\hrule
  \begin{flushleft}flushed left\end{flushleft}
  \begin{center}centered\end{center}
  \begin{flushright}flushed right\end{flushright}
\hrule
```

You can display text:

flushed left

centered

flushed right

7.2 Tables and Listings

Authorea supports figures as separate text blocks, so you should never have to write `\begin{figure}` in your documents. However, listings and tables are perfectly acceptable. In fact, you could be curious to find out the technique which we use here for aligning our listings and tables in two horizontal columns. The relevant snippet is:

```
\begin{figure}[h!]
\begin{minipage}{0.7\textwidth}
 ... left-hand-side content ...
\end{minipage}
\begin{minipage}{0.2\textwidth}
... right-hand-side content ...
\end{minipage}
\end{figure}
```

The reason we use `\begin{figure}[h!]` here is to explicitly order TEX to position the figure as close as possible to the current insertion point.

7.3 Note on Margins

If you are very serious about producing beautiful HTML and PDF, you will sooner or later need to understand how margins work and how they differ between the two paradigms. As long as you are not using advanced features, Authorea can automate all those considerations for you, but the moment you want beautiful side-by-side tables, you may be forced to become painfully aware of the PDF-related float widths and margin sizes.

Best to remember: If you don't want to know anything about margins and just want to disseminate your preprint to colleagues, use `\usepackage{fullpage}` which disables page margins. Otherwise, try to design your minipage environments so that they never sum up to over `0.9\textwidth` and ideally `0.8\textwidth`, to avoid unpleasant overflows. There are other TEX tricks one can employ but we recommend heavily **against** indulging in low-level fine-tuning, as it is both a time sink and a great potential source of frustration.

7.4 Note on Vertical Overflow

HTML pages are designed to be infinitely scrollable. However, this is clearly not the case in printable PDF documents, and if you want your tables to display nicely on a paper page you need to take precautions against vertical overflow early. A simple solution is to use the `{longtable}` environment instead of `{tabular}`, which will auto-break the pages for you, when exporting to PDF.

7.5 Note on Format-specific Directives

In few unfortunate cases the investment of writing format-independent LATEX simply isn't worth the result. For example, it may be very easy to align two `{minipage}` blocks next to each other in HTML, but extremely painfull to fit them on the page in PDF. To achieve a delightful outcome in both formats in such cases, it may be needed to resort to platform-specific directives. LaTeXML offers support for this distinction using its own conditional operator: `\iflatexml`.

Here is an example setup that you can add to your `header.tex` file:

```
\def\onlyHTML#1{\iflatexml #1\fi}
\def\onlyPDF#1{\iflatexml\else #1\fi}
```

and then use it to specify PDF-only line-breaks for your `{minipage}` blocks:

```
% ...
\end{minipage}\onlyPDF{\newline}
\begin{minipage}{% ...
```

The most common use of PDF-specific directives would likely remain the need for hard page breaks:

```
\onlyPDF{\newpage}
```

Last, but very importantly, please be reminded that at Authorea we strongly advise against attempting to do low-level tweaking of your document, both because of the numerous dragons lurking on that road, and because the Authorea editing experience is much more rewarding in both speed and writing enjoyment.

8 LaTeX Programming? Only if you really *really* have to...

8.1 Generating a multiplication table

To illustrate why you should ***avoid*** doing any low-level TeX programming yourself[2] here is an example of the hoops you need to jump through to typeset the multiplication table from five to nine:

[2] Unless you really know what you are doing, and you really must

```
% We need some variables to loop on, for which we utilize TeX's
    counters
\newcounter{a}
\newcounter{b}
\newcount\ab
% Store the output rows in a special token variable
%  (typesetting inside tabular = tricky)
\newtoks\allrows
% The darkest black magic happens here.
% It is needed to make \addrow update the \allrows variable
%   while also:
%   1) keeping it global and
%   2) fetching the actual values from the TeX counters.
% Doing this "the right way" is neither intuitive nor easy.
\newcommand\addrow[3]{%
  \begingroup\edef\container{\endgroup
    \noexpand\global\noexpand\allrows{\the\allrows
      #1 & #2 & #3\noexpand\\}}\container}
% Some simple macros for initializing and printing our tokens
    variable
\newcommand*\resetrows{\global\allrows{}}
\newcommand*\printrows{\the\allrows}
%
% We can now initialize and generate our table rows
\resetrows
% For A from 5 to 9
\setcounter{a}{4}
\loop\ifnum\thea<9
  \stepcounter{a}
  \setcounter{b}{0}
  % and B from 1 to 9
  {\loop \ifnum\theb<9
    \stepcounter{b}
    % No shock, \ab is a times b
    \ab=\thea
    \multiply\ab by \theb
    % Prepare the table row
    \addrow{\thea}{\theb}{\the\ab}
  % and loop
  \repeat}
  % We add a horizontal line to separate the different A blocks
  % notice that even doing that properly requires "black magic"
  % in order to expand \hline at the right place and time
  \expandafter\global\expandafter\allrows\expandafter{\the\allrows\
      hline}
\repeat
%
% Finally, show our work, by writing a table with minimal markup:
\begin{tabular}{|rrr|}
\hline $A$ & $B$ & $A\times B$ \\ \hline
  \printrows
\end{tabular}
```

A	B	$A \times B$
5	1	5
5	2	10
5	3	15
5	4	20
5	5	25
5	6	30
5	7	35
5	8	40
5	9	45
6	1	6
6	2	12
6	3	18
6	4	24
6	5	30
6	6	36
6	7	42
6	8	48
6	9	54
7	1	7
7	2	14
7	3	21
7	4	28
7	5	35
7	6	42
7	7	49
7	8	56
7	9	63
8	1	8
8	2	16
8	3	24
8	4	32
8	5	40
8	6	48
8	7	56
8	8	64
8	9	72
9	1	9
9	2	18
9	3	27
9	4	36
9	5	45
9	6	54
9	7	63
9	8	72
9	9	81

8.2 Generating a Christmass Caroll

In a classic argument about whether a naive approach to **counting the words** of a TEX manuscript could be successful, David Carlisle provided the following snippet as an example:

```
\let~\catcode~‘76~‘A13~‘F1~‘j00~‘P2jdefA71F~‘7113jdefPALLF
PA’’FwPA;;FPAZZFLaLPA//71F71iPAHHFLPAzzFenPASSFthP;A$$FevP
A@@FfPARR717273F737271P;ADDFRgniPAWW71FPATTFvePA**FstRsamP
AGGFRruoPAqq71.72.F717271PAYY7172F727171PA??Fi*LmPA&&71jfi
Fjfi71PAVVFjbigskipRPWGAUU71727374 75,76Fjpar71727375Djifx
:76jelse&U76jfiPLAKK7172F711727 1PAXX71FVLnOSeL71SLRyadR@oL
RrhC?yLRurtKFeLPFovPgaTLtReRomL;PABB71 72,73:Fjif.73.jelse
B73:jfiXF71PU71 72,73:PWs;AMM71F71diPAJJFRdriPAQQFRsreLPAI
I71Fo71dPA!!FRgiePBt’el@ lTLqdrYmu.Q.,Ke;vz vzLqpip.Q.,tz;
;Lql.IrsZ.eap,qn.i. i.eLlMaesLdRcna,;!;h htLqm.MRasZ.ilk,%
s$;z zLqs’.ansZ.Ymi,/sx ;LYegseZRyal,@i;@ TLRlogdLrDsW,@;G
LcYlaDLbJsW,SWXJW ree @rzchLhzsW,;WERcesInW qt.’oL.Rtrul;e
doTsW,Wk;Rri@stW aHAHHFndZPpqar.tridgeLinZpe.LtYer.W,:
```

Source: http://ctan.org/pkg/xii

Clearly, a naive approach may find a handful, if any words, as this TeX program is obfuscated to a degree of infamy. In fact, it is a demonstration that the only correct way of counting words in a LATEX document is by inspecting the final produced output, be it a PDF or HTML document. Authorea is one of the few tools that correctly approaches word count and can tackle this example successfully, with a click of a button.

You can enjoy the full carol here.