Glisterings

Peter Wilson

> Sound like bels, and shine like lanternes.
> Thunder in words and glister in works.
>
> *School of Abuse*, Stephen Gosson

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine.

> Something old, something new,
> Something borrowed, something blue.
>
> *Traditional: Advice to the Bride*

This issue's column reiterates two items from the electronic TEXMAG journal, years ago.

> I didn't go to the moon, I went much further — for time is the longest distance between two places.
>
> *The Glass Menagerie*, Tennessee Williams

1 Timelines

This is a slightly edited version of an article that Don Hosek wrote for TEXMAG, titled *Timelines with plain TEX and LATEX* [1].

In one issue of *TUGboat* (Vol. 8, No. 2), there was a query for a macro to draw timelines in TEX. At the time, I had just finished writing DVIview and was waiting for bugs to surface and my paycheck to arrive with little else to do, so I decided to tackle the problem.

To make the problem more interesting, I decided to make the macro work in both LATEX *and* plain TEX. A sample input file should look something like:

```
%%% LaTeX sample
\documentclass{article}
\usepackage{timeline}
\def\TeXMaG{\TeX
   M\kern-.1667em\lower.5ex\hbox{A}%
    \kern-.2267emG}
\begin{document}
This is a timeline of the history of
the first year of \TeXMaG.
\begin{timeline}{2in}(0,180)
\optrule
\item[12]{Jan. 24}{No. 1}
\item[33]{Mar. 6}{No. 2}
\item[43]{Mar. 25}{No. 3}
\item[81]{May. 13}{No. 4}
\item[102]{Jun. 25}{No. 5}
\item[132]{Aug. 24}{No. 6}
\item[160]{Oct. 10}{No. 7}
\item[179]{Dec. 31}{To be}
\end{timeline}%% Must be a comment here!!!
If text immediately follows the end of the
timeline then a comment is required
otherwise there is an extraneous space at the
start of the text. A blank line following the
end acts normally, whether or not there has
been a comment.
\end{document}
```

And something similar in plain TEX. In the example above, I used sort keys to control the spacing between entries. I also could have had a timeline whose entries looked like this:

```
\begin{timeline}{1.5in}(1750,1900)
\optrule
\item{1773}{The Tea Party}
\item{1812}{War of 1812}
\item{1849}{Gold rush of '49}
\item[1862]{1862--5}{Civil War}
\item{1876}{Little Big Horn} % added by PW
\end{timeline}
```

where the dates themselves control the placement. Note that the entry for the Civil War uses a sort key to allow the year to be 1862–5. This was a pretty big problem. The comments in the code below give a fair idea of how to *use* the macro; the remainder of this section will deal with how the macros themselves *work*.

First of all, it helps to have some idea of how `\begin{...}` and `\end{...}` work in LATEX. To view it in a simplified form, when the command `\begin{FOO}` is invoked, LATEX issues the commands `\begingroup` followed by `\FOO`; similarly, `\end{FOO}` issues the commands `\endFOO` followed by `\endgroup`. Therefore, to allow an environment to function in plain TEX, all we need to do is include an extra grouping with `\FOO...\endFOO` and provide copies of any LATEX internal macros used by the environment. Both of these tasks are fairly simple, and in the timeline macros, the only LATEX internal macro called is `\@ifnextchar` (this is a very handy macro for many reasons, and gives some insight into the mysteries of LATEX).

`\@ifnextchar` is called with the general form:

```
\@ifnextchar X{YES}{NO}
```

The timeline macros use this for `\item` to check to see if the optional argument (enclosed in `[]`s) is present. If the next character after `\@ifnextchar` matches `X` (`X` cannot be a space) then `YES` is executed, otherwise `NO` is executed. In the specific case here, this is done with the call:

```
\@ifnextchar[\@item\@itemnosrtkey
```

This calls `\@item` if the optional argument is present, and `\@itemnosrtkey` if it isn't. In addition, the character that is tested remains in the input stream, so `\@item` has a parameter list that looks like
`\@item[#1]#2#3`
rather than
`\@item#1]#2#3`.

The definition of `\item` for the timeline macros is kept local so it won't interfere with other uses of that control sequence name by either plain or LaTeX. The main work of this is done by `\@item`, which takes three arguments: the first argument is used to determine the vertical placement of the timeline item, the second argument is the nominal date and the third a description. `\@itemnosrtkey` calls `\@item` using the nominal date as the first argument as well.

The placement of the item on the timeline is determined by taking the date number (first parameter) and converting to a number between 0 and the length of the timeline as specified with the arguments to the `\timeline` (`\begin{timeline}`) macro. This number is then multiplied by 1/65536 times the length of the timeline as specified by the user. The factor of 1/65536 prevents an arithmetic overflow from occuring at the cost of reducing accuracy (measurements are only kept accurate to one point). Finally, this number is divided by the length of the range of date number values and then multiplied by 65536 which gives a dimension specifying how far down from the top of the timeline the entry should be placed.

The actual placement is accomplished with the `\dlap` macro from the toolbox of TeXMaG Vol. 1 No. 3 [by Barbara Beeton]. By placing the necessary text after a vertical `\kern`, inside a vertical lap, we are able to print information anywhere on the timeline without changing our vertical position. This does have the disadvantage of using a lot of box memory and may run into problems with very complicated timelines, but it seemed like a good idea at the time.

The final interesting facet of the macros is the (simple) way that two entries that are close together are resolved. After an entry is printed, the vertical dimension specifying its placement is stored in the dimen register `\itwashere`. When the next entry is to be printed, the current vertical placement is compared to `\itwashere`; if the difference is less than 12pt, and the entry would normally be placed on the left, then the entry is printed on the right. Otherwise it is printed on the left. This algorithim works well for two closely placed entries but fails for three closely placed entries (the two on the left will likely overlap).

This is a timeline of the history of the first year of TeXMaG.

Jan. 24 No. 1 •
Mar. 6 No. 2 •
• **Mar. 25** No. 3
May. 13 No. 4 •
Jun. 25 No. 5 •
Aug. 24 No. 6 •
Oct. 10 No. 7 •
Dec. 31 To be •

If text immediately follows the end of the timeline then a comment is required otherwise there is an extraneous space at the start of the text. A blank line following the end acts normally, whether or not there has been a comment.

Figure 1: First timeline

The macros presented work for simple timelines, but probably will be deficient for more complex timelines. Hopefully, this explanation of the macros will help in customizing them for your own purpose, or in writing a timeline macro of your own.

```
%%% File: timeline.sty
%%% Works with either LaTeX or plain TeX
%%%
%%% In LaTeX:
%%% \begin{timeline}{length}(start,stop)
%%%   . . .
%%% \end{timeline}
%%%
%%% in plain TeX
%%% \timeline{length}(start,stop)
%%%   . . .
%%% \endtimeline
%%% in between the two, we may have:
%%% \item{date}{description}
%%% \item[sortkey]{date}{description}
%%% \optrule
%%%
%%% the options to timeline are:
%%%     length --- The amount of vertical space
%%%        that the timeline should use.
%%%     (start,stop) --- indicate the range of
%%%        the timeline. All dates or sortkeys
%%%        should lie in the range [start,stop]
%%%
%%% \item without the sort key expects date to
%%%       be a number (such as a year).
%%% \item with the sort key expects the sort
%%%       key to be a number; date can be
%%%       anything. This can be used for log
%%%       scale timelines or dates that
%%%       include months or days.
```

```
%%% putting \optrule inside of the timeline
%%%       environment will cause a vertical
%%%       rule to be drawn down the center
%%%       of the timeline.

\catcode`\@=11     % Pretend @ is a letter
\newcount\startat    \newcount\tllength
\newdimen\putithere  \newdimen\itwasthere
\newcount\scr@tchi   \newdimen\scr@tchii

% A vertically centered lap
\long\def\ylap#1{\vbox to \z@{\vss#1\vss}}

% Vertical 'laps'; cf. \llap and \rlap
\long\def\ulap#1{\vbox to \z@{\vss#1}}
\long\def\dlap#1{\vbox to \z@{#1\vss}}

\def\timeline#1(#2,#3){%
  \ifvmode\else\par\fi$$\vbox to#1\bgroup
          % The \vbox command is
          % surrounded by $$..$$ to make it
          % fit in well with paragraphs.
  \offinterlineskip
  \startat=#2\tllength=#3
    \advance\tllength by-\startat
% \tllength should be the total length of
%  the timeline.
  \def\item{\@ifnextchar[\@item\@itemnosrtkey}
  \def\@item[##1]##2##3{\scr@tchi=##1
    \advance\scr@tchi by-\startat
    \putithere=#1
    \divide\putithere by 65536 % avoid overflow
% only remain accurate to 1pt in the
% next set of calculations
    \multiply\putithere by \scr@tchi
    \divide\putithere by\tllength
    \multiply\putithere by 65536
% Now \putithere has how far
% down we should go for this item.
    \scr@tchii=\putithere
    \advance\scr@tchii by -\itwasthere
    \ifdim\scr@tchii<12pt
      \ifx\lrswitch L
        \@putright{\putithere}{##2}{##3}
      \else
        \@putleft{\putithere}{##2}{##3}
      \fi
    \else
      \@putleft{\putithere}{##2}{##3}
    \fi
    \itwasthere=\putithere}
  \def\@itemnosrtkey##1##2{%
    \@item[##1]{##1}{##2}}
  \def\@putright##1##2##3{\dlap
    {\kern##1\centerline
      {\rlap
        {\ $\bullet$\hskip1.5em{\bf ##2}
         \ ##3}}}
    \let\lrswitch=R}
  \def\@putleft##1##2##3{\dlap
    {\kern##1\centerline
      {\llap
        {{\bf ##2} \ ##3\hskip1.5em$\bullet$
          \ }}}
    \let\lrswitch=L}
  \def\optrule{\dlap
    {\centerline
% This calculation is kept local
      {\dimen0=#1 \advance\dimen0 by 6pt
        \vrule depth \dimen0 height-6pt}}}

% Put the extra \vskip in a \vbox to hide
% it from the math gods.
\def\endtimeline{%
  \vfill\egroup\vbox{\vskip\baselineskip}$$}
\ifx\@latexerr\undefined
  \def\@ifnextchar#1#2#3{%
    \let\@tempe #1
    \def\@tempa{#2}\def\@tempb{#3}
    \futurelet\@tempc\@ifnch}
  \def\@ifnch{%
    \ifx \@tempc \@sptoken
      \let\@tempd\@xifnch
    \else
      \ifx \@tempc \@tempe
        \let\@tempd\@tempa
      \else
        \let\@tempd\@tempb
      \fi
    \fi \@tempd}
% NOTE: the following hacking must precede
%       the definition of \: as math medium
%       space.
% make \@sptoken a space token
  \def\:{\let\@sptoken= } \:
  \def\:{\@xifnch}
  \expandafter\def\: {\futurelet\@tempc\@ifnch}
\catcode`\@=12 % Stop pretending @ is a letter
\fi
\endinput
```

Using the above code, the result of the initial example is in Figure 1 and the second is in Figure 2.

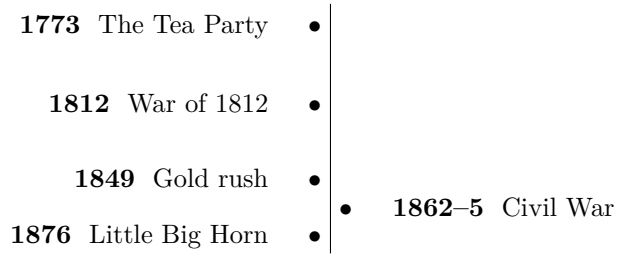

Figure 2: Second timeline

> Gaul as a whole is divided into three parts.
>
> *De Bello Gallico*, Julius Caesar

2 Parsing a filename

This is another (slightly edited) article from a later issue of TEXMAG [2].

Sometimes it is nice to be able to use the information in a `filename.tex` as information in a particular document. For example, suppose I wanted to typeset TEXMAG on real paper, and be able to have the volume and issue numbers read from the title of the file that TEX was processing, and subsequently assigned to tokens for use in the document, perhaps in a header. Say my file was named `TEXMAG-5-1.TEX`. The following would isolate the 5 and the 1 for use within the TEX document:

```
% This particular idea was developed by our
% chief consultant Dr. John McClain

\newtoks \volumenumber
\newtoks \issuenumber

\def\parse#1-#2-#3-{\global\volumenumber={#2}
                    \global\issuenumber={#3}}

\expandafter\parse\jobname-

%%% for a TeX headline
\headline={Volume \the\volumenumber,
            Number \the\issuenumber
\hfil page \folio}
% end of macro
```

Notice the `\jobname` contains the name of the file (without any extension, see *The TEXbook*, p. 213). The `\expandafter` allows you to piece apart this token into its volume and number. We also had to chose a special delimiter which would conform to filename standards and be a legal parameter delimiter in TEX. A space would not have worked as a legal file name. A hyphen was our best choice. When you test this, remember that the filename must conform to the parameter specs of `\parse` (in this case, two hyphens, i.e., `XXXX-N-N.TEX`).

The essence of the code in the TEXMAG article is the `\parse` macro. The `\jobname` of the document you are now reading is '`tb103glister`', which does not match the requirements of `\parse`. The following code demonstrates that macros based on `\parse` can work with names other than `\jobname`, provided that they expand into the expected format.

For instance:

```
\newcommand*{\jname}{glisten-n16-v3.tex}
\newtoks \pwfirstsub
\newtoks \pwsecondsub
\def\parse#1-#2-#3.#4-{%
    \global\pwfirstsub={#2}
    \global\pwsecondsub={#3}}
\newcommand*{\parsit}[1]{\expandafter\parse#1-}

\verb?\parsit{\jname}? \parsit{\jname}
  File \jname\ with: \\
Number \the\pwfirstsub,
and Version \the\pwsecondsub.
```

And the result of the above code is:

`\parsit{\jname}` File glisten-n16-v3.tex with: Number n16, and Version v3.

The basic idea of `\parse` can be applied to any multipart string that has well-defined delimiters between the parts.

References

[1] Don Hosek. Timelines with plain TEX and LATEX. *TEXMAG*, 1(7), October 1987. `http://mirror.ctan.org/digests/tex-mag/v1.n7`.

[2] John McClain. The toolbox. TEXMAG. `http://mirror.ctan.org/digests/tex-mag/v5.n1`.

◇ Peter Wilson
12 Sovereign Close
Kenilworth, CV8 1SQ
UK
herries dot press (at)
earthlink dot net