TEXniques

DTP with TEX

Roland Kwee
S.S.R. Kwee Computer Consultancy
Amsterdam
email: `rolandkwee@acm.org`

abstract
A set of simple macros is presented, in the style of modular programming, to make it easy to put texts at arbitrary positions on the page, without being restricted by formatting rules. This is not only useful for single-page documents like announcements and business cards, but also for designing stationery with letterheads or printing labels. The method is based on putting kerned texts in boxes with zero horizontal and vertical size and staying in vertical mode. This works with plain, and any other, TEX, and can be combined with any other document formatting.

keywords
programming, DTP, vertical mode

Introduction

TEX is designed to “write beautiful books” and is therefore usually run with a carefully designed set of formatting rules. The idea is that an author should be concerned with writing down content while leaving typography to the publisher. By separating content and typography it is easy to create large documents with a uniform and consistent appearance.

Using the predefined plainTEX or LATEX styles, it is easy to fill a document with content. Changing a style or format, or even creating a new one, is much harder: you need to do some TEX programming. This isn’t anything trivial, as can be guessed from the typical TEX appearance of documents typeset with esp. LATEX.

Personally, I can live with the formatting that others made available. What I do want, however, is to personalize the appearance of my correspondence, i.e., I want to design my own letterhead. This involves more than just adding a page headline or footline.

Another problem is that formatting gets in the way of designing very simple documents consisting of a single page. For a flyer, a business card, address labels etc, line breaking rules and page breaking rules are irrelevant. Worse, all these formatting rules make it hard to put a text a little higher or lower, next to that other text, etc. In fact, in such cases we don’t want the carefully designed typography for large documents. We want to explicitly break all the rules and to be able to put the texts at any arbitrary position on the page.

Traditionally, there is a big schism between desktop publishing (DTP) programs that allow you unlimited freedom for the placement of texts, and text processing programs like TEX that place your texts according to rigorous built-in typography restrictions. However, being a TEX proponent (bigot?), I also want to fill my DTP needs with TEX. This is not just to save the money for a DTP program license, but also because some of my documents, notably letters, need the capability of TEX for the body of the text, combined with DTP capabilities for the letterhead.

Programming obstacles

Over the years I have done a lot of TEX programming, varying from specifying a new page size, e.g., A4 instead of the default U.S. letter size of 8.5 by 11 inches, to writing macros, to making a “typography” to print a nice calendar from the output of a program written in C. Each time I had to fix lots of problems with horizontal and vertical mode, unwanted spaces, adjusting glue, and so on. I can program in many languages, but programming in TEX still has the most “magic”. I must admit that I could solve practically all problems by careful reading of “The TEXbook”. And it is clear that all those difficulties are somehow unavoidable when programming text with a program written with text. Some text is text, and other text is program, all in one file, even in one line or word.

Another, unrelated, problem is that it is still common practice to write TEX programs in “optimized” style, that is, optimized for the computer, not the human. Looking in a format or style file, or at the examples in “The TEXbook” makes me think of a Forth or Perl program, or a Postscript file, also called “write-only” files. The effect is that TEX programming is discouraged and restricted to the high class of TEXperts and TEXnicians.

Some programming that works with plain TEX does not work with LATEX, and vice versa. The reason is that LATEX does not only add high-level features like nice table commands and indexing, but also introduces new programming concepts like variables or lengths, different kinds of boxes, and all that crap. Lamport says in his LATEX reference manual *There is no easy way to tell whether a Plain TEX command will cause trouble, except by trying it.*

Keep it easy

According to Albert Einstein, we should *keep it as simple as possible* (though not simpler). He was much more clever than I am, so I stick with this. How can we make programming in TEX easy?

Here are my recommendations:

□ *Program in plain TEX.* Knuth's book is an excellent reference for plain TEX programming. For LATEX programming you'd have to study the format file. Good Luck.

□ *Stay in vertical mode.* A TEX document always starts in vertical mode. Basically, you leave vertical mode when you start a new paragraph, or by an `\indent` or `\noindent`. Hence, all macros defined before the first paragraph are in vertical mode. If you stay in vertical mode, you don't have to worry about getting an unwanted space, or having to end lines with a `% comment`. It is essential to avoid unwanted spaces, if you want to have precise control of the placement of a text on the page. An unwanted space means glue, need I say more?

□ *A macro should only do one simple thing.* This should not need further explanation. For those who need some: this is to enable reuse of the macro instead of reinventing the wheel.

□ *Combine simple macros to achieve complicated goals.* This is also called breaking down a complicated problem into several simpler problems. TEX programming is difficult enough, so even simple problems can be challenging. Personally, I would not even try to solve complicated problems directly with TEX programming.

□ *Group related macros into a module.* As several simple macros are needed to solve a real world problem, such macros should be grouped and kept together as a module. All globally visible names should begin with the name of the module. All macros should be in a file with the name of the module, and nothing more should be in that file. This way, a document can safely `\input` a module without naming conflicts (assuming you name the modules suitably).

□ *Add sufficient documentation.* As a minimum, a file should begin with a comment describing its purpose. Each macro should have a description of its function, parameters and usage. Each trick should have ample explanation.

□ *Use indentation.* There is really no justification for having to guess the level of parenthesation in effect at a word in a macro.

Easy macros

The task with DTP is to put a text on a particular position on the page. The macro DTPVPOS does just this. Actually, there is only one macro of one line in this DTP package:

```
% file: dtp.tex
% purpose: DTP capability for TeX

% V P O S
%
% Returns a vbox with zero dimensions,
% containing a word or vbox #3
% at offset (x=#1, y=#2).
% See: The TeXBook, appendix D, p. 389.
%
\def\dtpvpos#1#2#3{
% Place word at (x,y). #1=xpos #2=ypos #3=word
  \vbox to0mm{\kern#2\hbox{\kern#1{#3}}\vss}%
  \nointerlineskip
}
```

It takes three parameters: a text, an X and a Y coordinate. The text can be a `\vbox`.

The macro returns a `\vbox` of size zero. It is assumed that the macro is called when TEX is still at the starting position at the top-left corner of the document. After the macro call, TEX has not moved a bit, so subsequent calls to this macro can be made.

In the returned box the input box is kerned horizontally and vertically from the top-left corner to the specified position.

The macro uses a Dirty Trick from "The TEXbook", so I don't need to explain here why `\nointerlineskip` is used. Therefore I did not apply the indentation recommendation here.

The macro could also have been written with an hbox, using `\rlap`. This is left as an exercise for the masochist, because it is much better to work in vertical mode as explained before.

The description of the macro isn't really complete without an example of its use. Here is the start of the design of a pamphlet to announce a concert.

```
% Concert announcement of
% the ensemble ''encore''.
% Status: unfinished.

\input dtp

\parindent=50mm
% indentation of first line of a paragraph

\font\fa=pctu8r at 30pt
```

```
% pctu8r=Cheltenham-Ultra

% Define the musical part of the announcement
\def\vhead{
  \vbox{
    \dtpvpos{0mm}{0mm}{\fa Encore}
    \dtpvpos{0mm}{10mm}{%
       o.l.v. Ren\'e de Grote}
    % Put more texts here
  }
}

% Put together the various parts of
% the announcements
% C programmers would call this
% the ‘‘main’’ macro.
\def\vconcert{
  \vbox{
    \dtpvpos{20mm}{20mm}\vhead
    %\dtpvpos{0mm}{200mm}\vsponsor
    %to be defined
  }
}

% Execute the macros
\vconcert

\bye
```

It shows that it is easy to put a text of any font and any size, an important DTP feature, at any place on the page.

When the pamphlet is finished, it would probably consist of a few top-level macros, like one for the name of the ensemble, the date and location, and another for the name and logo of the sponsors. Each of these macros would contain calls to DTPVPOS for the various text elements. A “main” macro would then call DTPVPOS to place the text blocks of each top-level macro to the proper places on the page. This way, a pamphlet can be designed hierarchically.

All the while normal TEX formatting can be applied either within a box that is passed to a DTPVPOS call, or after the last DTPVPOS call for non-DTP text. The latter method can be used for a letterhead.

Another “recursive” application is label printing. First, make a macro calling DTPVPOS several times to construct one label with its several pieces of text, like name, street, city, logo. Then, make a macro calling DTPVPOS several times, one for each label on the sheet, with the first macro as the text argument. If you have ever tried to make an `\output` routine for two columns of five labels on a sheet, you will find the DTP method much easier. It will work the first time. And it will be easy even if the labels are non-uniformly distributed over the sheet.

Conclusion

TEX programming can be easy, and still powerful, by applying commonsense programming techniques. The dirty tricks can be hidden in a macro. TEX can be used both for traditional consistent document formatting and for traditional non-consistent desktop-publishing-like page formatting.

It is hoped that this style of macro programming will be used in other TEX applications, so that I can understand those. I even hope that this will make the use of, and programming in, TEX more available to “the masses”, and that we can use TEX also to create *masterpieces of the desktop publishing art!*