Exact layout with LaTeX

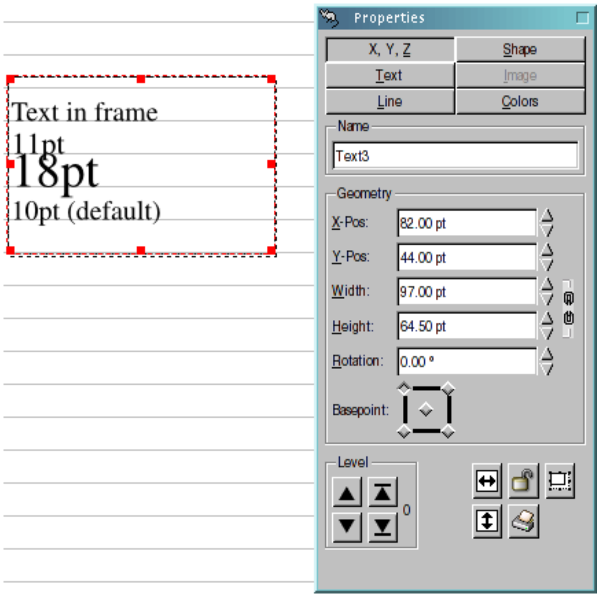
Implementing a letterhead

Keywords
letterhead, picture environment, boxes, PostScript, docstrip

Abstract
This article describes several techniques useful for implementing a professionally-designed layout such as a letterhead.

The Economics Department of Groningen University recently got new logos and an updated house style, including a new letter template.

The LaTeX implementation of the old version was done by Erik Frambach. Although I didn't have to change the basic structure very much, it still involved more than just changing some parameters. This article highlights some of the tricks, both new and inherited, that were used.



Name: Text3
X-Pos: 82.00 pt
Y-Pos: 44.00 pt
Width: 97.00 pt
Height: 64.50 pt
Rotation: 0.00 °

The page layout perspective

The previous time, the designer kept pointing out how much simpler and more natural things were with QuarkXPress, and he did have a point.

In case you don't realize what you are missing: the screenshot on this page shows how you can place text in a page layout application[1].

The dialog box contains entries X-Pos and Y-Pos for exact page coordinates of the box on the left. This is very different from TeX and wordprocessors, where positioning is usually relative to the text cursor, and positioning relative to the page requires frightening contortions.

Grid typesetting

More advanced page layout applications support typesetting on a grid of evenly spaced horizontal lines. If things don't fit within their allotted space then either that fact is ignored or the text jumps to the next gridline. The Scribus screenshot illustrates the former: the '18pt' line only occupies the default 12-point line height, and tangles with the line above.

Our letterhead also uses a 12-point vertical grid, although the grid is enforced only for the letterhead itself; TeX and LaTeX haven't been designed with grid typesetting in mind, and I know of no simple, reliable way to enforce grid typesetting in general with LaTeX.

Units

Most of the graphics industry works with 'big points', of which there are exactly 72 to the inch. The specified line height was 12bp. It seemed best to lift all fontsize definitions from size10.clo and modify them for big points.

I found TeXCalc[2], a utility from Taco Hoekwater for unit conversions, very useful.

Below, I'll mostly use big points for vertical measurements.

Using the picture environment

If you need to place items at specified positions, then the LaTeX picture environment comes to mind. This is how you use it:

```
\unitlength=1bp
\begin{picture}(..,..)(..,..)
picture commands
\end{picture}
```

The first pair of numbers is the size of the picture, in unit lengths; the second pair is optional and indicates the coordinates of the lower left corner, which need not be (0,0).

Actually, the size is not necessarily the real size; it is the size that LaTEX is going to reserve for the picture. Therefore you can put in some extra whitespace by telling LaTEX that the picture is larger than it really is. You can also do the opposite: you can place picture objects at coordinates far outside the declared picture dimensions and LaTEX will typeset the rest of the page as if those objects aren't there.

There were two problems with the picture environment which had to be worked around:

□ The makebox command of the picture environment doesn't bother about baselines. In the picture below, we would like the descender of 'p' to extend below the baseline, but instead it sits on it:

```
\unitlength=1bp
\begin{picture}(12,15)(0,-5)
  \put(0,0){\line(1,0){12}}
  \put(0,0){\makebox(0,0)[bl]{p}}
\end{picture}
```

produces: p__

□ A lot of syntax can't be used inside the picture environment. As a workaround, you can prepare material in advance.

Plain TEX boxes

LaTEX itself has various box commands, such as `\mbox`, `\parbox` and the `minipage` environment. However, working with boxes can become incredibly verbose and roundabout if you insist on using the LaTEX box commands. So here comes a quick introduction to plain TEX boxes, at least of what I am going to use of them[3].

TEX uses box registers, which have to be allocated, *e.g.*:

```
\newbox{\pbox}
```

You can fill and place this box:

```
\setbox\pbox=... % define box contents
\copy\pbox %place the box contents
\box\pbox % place and clear the box
```

There are hboxes, vboxes and vtops. They are different in what you can put into them, not in how you can use them. hboxes are like LaTEX mboxes, but you use different syntax to fill them. `\\` does nothing inside an hbox. vboxes and vtops are different: when you start filling such a box you are in internal vertical mode, and `\\` does work. I am not going into the intricacies of modes in boxes; if you want to put anything complicated inside, then the LaTEX alternatives may be more appropriate.

Each box has a reference point, which is normally on a baseline at the left edge. For an hbox there is only one baseline. Vtop and vbox boxes can have more baselines. The reference point for a vtop it is the top baseline and for a vbox the bottom one.

From what we saw in the above example, for placement purposes it seems smart to pretend that there is nothing below the reference point. To this end, we use the `\smash` macro, which pretends that depth is zero, and width and height too.

The following example shows how this works out inside a picture environment.

```
% first, turn off parindent
\newlength\parsave
\parsave=\parindent
\parindent=0bp

\setbox\pbox=\hbox{p\\q}%
\setbox\qbox=\vbox{p\\q}%
\setbox\rbox=\vtop{p\\q}%

% boxes ready, can turn parindent back on
\parindent=\parsave

\noindent
\unitlength=1bp
\begin{picture}(40,40)(0,-15)
  \put(0,0){\line(1,0){40}}
  \put(0,0){\makebox(0,0)[bl]%
    {\smash{\box\pbox}}}
  \put(20,0){\makebox(0,0)[bl]%
    {\smash{\box\qbox}}}
  \put(30,0){\makebox(0,0)[bl]%
    {\smash{\box\rbox}}}
\end{picture}
```

gives you correct alignment with the baseline:

```
      p
pq    q p
        q
```

Since we can place objects freely anyway inside a picture environment, the choice between `\vtop` and `\vbox` is often a matter of taste. But if you do want a text block to start at a given point and the number of lines may vary, then a `\vtop` box is the way to go. The relevant part of the classfile might look as follows:

```
\def\@toname{}
\def\toname#1{\def\@toname{#1}}

\long\def\@toaddress{}
\long\def\toaddress#1{%
  \long\def\@toaddress{#1}}
```

```
\def\@frominfo{%
  This Isme \\
  MyStreet 99 \\
  9999 ZZ MyCity \\
  Phone 0123456789}

\newbox\frombox
\newbox\logobox
\newbox\tobox
\def\makeletterhead{%
  \setbox\frombox=\vtop{\@frominfo}%
  \setbox\logobox=\hbox{%
    \includegraphics[width=1in]{logo}}%
  \setbox\tobox=\vtop{\@toname \\\@toaddress}%
  \unitlength=1bp
  \begin{picture}(300,126)
    \put(280,192){\makebox(0,0)[bl]%
      {\smash{\box\logobox}}}
    \put(280,180){\makebox(0,0)[bl]%
      {\smash{\box\frombox}}}
    \put(0,180){\makebox(0,0)[bl]%
      {\smash{\box\tobox}}}
  \end{picture}}
```

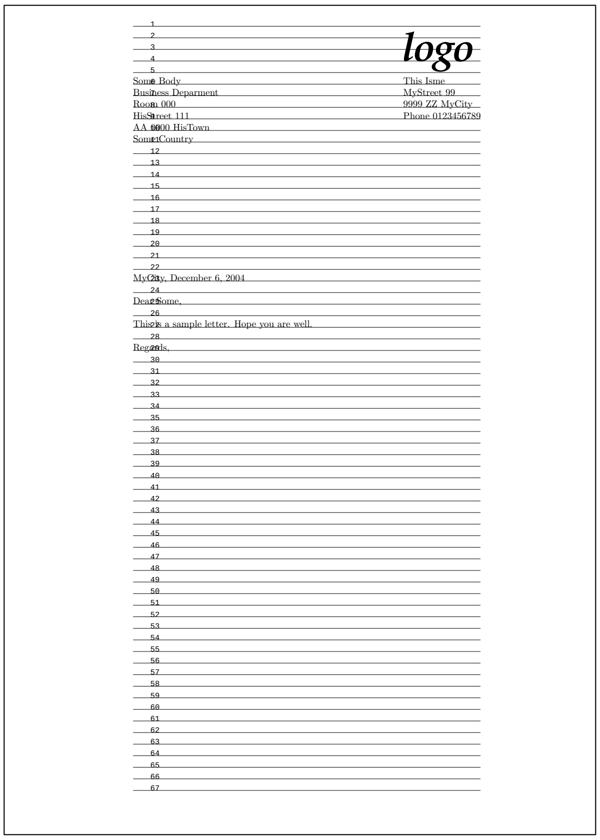
You can use this classfile as follows:

```
\pdfoptionpdfminorversion=3
\documentclass{letterdemo}
\begin{document}
\toname{Some Body}
\toaddress{%
  Business Deparment \\
  Room 000 \\
  HisStreet 111\\AA 0000 HisTown \\
  Some Country}
\makeletterhead

MyCity, \today

Dear Some,

This is a sample letter. Hope you are well.

Regards,
\end{document}
```

For a full listing of the classfile, see the end of the article.

Printing a grid

In order to check placement visually, you can print a grid as part of the page header; see Figure 1. The file grid.eps is hand-written PostScript:



logo

Some Body
Business Deparment
Room 000
HisStreet 111
AA 0000 HisTown
Some Country

This Isme
MyStreet 99
9999 ZZ MyCity
Phone 0123456789

MyCity, December 6, 2004

Dear Some,

This is a sample letter. Hope you are well.

Regards,

Figure 1. The letter printed with a grid

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 360 806
<< /PageSize [360 806] >> setpagedevice
0.4 setlinewidth
/Courier findfont 8 scalefont setfont
1 12 793 {
 newpath
 dup 0 exch moveto
 dup 360 exch lineto stroke
 dup 18 exch moveto
 dup 12 idiv 67 exch sub 2 string cvs show
} for
```

The nice thing of PostScript is that you can draw lines simply with commands `moveto`, `lineto` and `stroke`, using absolute page coordinates. This basic simplicity gets somewhat obfuscated by PostScript's lack of syntactic niceties, necessitating some juggling with `dup` and `exch`. So you may prefer to do the same with TEX macros.

Ghostscript will convert this to grid.pdf with the right boundingbox, thanks to the page definition on the third line. Without this line, you need a script such as epstopdf for conversion.

You can place it on the page as part of the page header code:

```
\def\ps@debug{%
  \def\@oddhead{%
    \smash{\raisebox{-705bp}%
      {\includegraphics{grid}}}}%
    \let\@oddfoot\@empty}
\pagestyle{debug}
```

If you can't sort out the exact value of the first raisebox parameter then just use trial and error.

Removing debug code with docstrip

In this simplified example, there is only one piece of code that needs removing. In more complex cases, you can mark the debug code

```
%<*debug>
...
%</debug>
```

and use a docscript 'batchfile' `myletter.ins` to remove them:

```
\input docstrip
\generate{\file{myletter.cls}%
  {\from{letterdemo.cls}{!debug}}}
\endbatchfile
```

Run this batchfile as follows

```
\latex myletter.ins
```

to get a version `myletter.cls` of your classfile without debug code.

The full listing

```
\LoadClass[a4paper]{article}
\usepackage{graphicx}

% duplicate definition of normalsize from size10.clo, but
% use big points for normal line spacing
\renewcommand\normalsize{%
   \@setfontsize\normalsize{10pt}{12bp}
   \abovedisplayskip 10\p@ \@plus2\p@ \@minus5\p@
   \abovedisplayshortskip \z@ \@plus3\p@
   \belowdisplayshortskip 6\p@ \@plus3\p@ \@minus3\p@
   \belowdisplayskip \abovedisplayskip
   \let\@listi\@listI}
\normalsize
\textwidth=360bp

\parindent=0bp
\parskip=12bp

\pagestyle{empty}
%<*debug>
\def\ps@debug{%
  \def\@oddhead{%
    \smash{\raisebox{-705bp}{\includegraphics{grid}}}}%
    \let\@oddfoot\@empty}
\pagestyle{debug}
%</debug>

\def\@toname{}
\def\toname#1{\def\@toname{#1}}

\long\def\@toaddress{}
\long\def\toaddress#1{\long\def\@toaddress{#1}}

\def\@frominfo{%
  This Isme \\
  MyStreet 99 \\
  9999 ZZ MyCity \\
  Phone 0123456789}

\newbox\frombox
\newbox\logobox
\newbox\tobox
\def\makeletterhead{%
  \setbox\frombox=\vtop{\@frominfo}%
 \setbox\logobox=\hbox{\includegraphics[width=1in]{logo}}%
  \setbox\tobox=\vtop{\@toname \\\@toaddress}%
  \unitlength=1bp
  \begin{picture}(300,127)
    \put(280,192){\makebox(0,0)[bl]{\smash{\box\logobox}}}
    \put(280,180){\makebox(0,0)[bl]{\smash{\box\frombox}}}
    \put(0,180){\makebox(0,0)[bl]{\smash{\box\tobox}}}
  \end{picture}}
```

Notes

1. Scribus, to be precise, Linux' answer to QuarkXPress and InDesign. Url: `http://ahnews.music.salford.ac.uk/scribus/`
2. Available from `http://tex.aanhet.net/utils/`. It is written in Perl/Tk.
3. A more complete discussion can be found in Victor Eijkhout's *TEX by topic* book, which can be downloaded for free from `http://www.eijkhout.net/tbt/`

Siep Kroonenberg
siepo@cybercomm.nl