Bijlage 35
Some funny macro's

Hans Hagen
pragma@pi.net

Keywords
dropped caps, CONTEXT

abstract
Sometimes documents can be enhanced with special typography for first characters or lines of chapters. In this article I present some macros for dropped caps and first line treatment. Although more advanced solutions are possible, the examples show at least how things work. Users can derive their own macros from them.

```
1  \unprotect
```

This module implements some typographics tricks that can be fun when designing document layouts. The examples use macros that are typical to CONTEXT, but non CONTEXT users can use the drop caps and first line treatment macros without problems. This module will be extended when the need for more of such tricks arises.

```
2  \ifx \undefined \writestatus \input supp-mis.tex \relax \fi
3  \writestatus{loading}{Context Support Macros / Fun Stuff}
```

Let's start with dropped caps, those blown up first characters of a paragraph. It's hard to implement a general mechanism that suits all situations, but dropped caps are so seldomly used that we can permit ourselves a rather user unfriendly implementation. \DroppedCaps

```
\DroppedCaps
  {\color[green]} {cmbx12}
  {2.2\baselineskip} {2pt} {\baselineskip} {2}
  Let's start
```

As we will see, there are 7 different settings involved. The first argument takes a command that is used to do whatever fancy things we want to do, but normally this one will be empty. The second argument takes the font. Because we're dealing with something very typographic, there is no real reason to adopt complicated font switching schemes, a mere name will do. Font encodings can bring no harm, because the alphanumeric characters are nearly always located at their natural position in the encoding vector.

This simple case shows us what happens when we apply minimal values. Here we used:

```
\DroppedCaps
  {\color[red]} {cmbx12}
  {\baselineskip} {0pt} {0pt} {1}
  This simple
```

In this ugly example the third argument tells this macro that we want a dropped capital scaled to the baseline distance. The two zero point arguments are the horizontal and vertical offsets and the last arguments determines the hanging indentation. In this paragraph we set the height to two times the baselinedistance and use two hanging lines:

```
\DroppedCaps
  {\color[red]} {cmbx12}
  {2\baselineskip} {0pt} {\baselineskip} {2}
  In this ugly
```

Here, the first character is moved down one baseline. Here we also see why the horizontal offset is important. The first example (showing the L) sets this to a few points and also used a slightly larger height.

Of course common users (typist) are not supposed to see this kind of fuzzy definitions, but fortunately TEX permits us to hide them in macros. Using a macro also enables us to garantee consistency throughout the document:

```
\def\MyDroppedCaps%
  {\DroppedCaps
     {\color[green]} {cmbx12}
     {5\baselineskip} {3pt} {3\baselineskip} {4}}

\MyDroppedCaps The implementation
```

The implementation of the general macro is rather simple and only depends on the arguments given and the dimensions of the strut box. We explicitly load the font, which is no problem because TEX does not load a font twice. We could have combined some arguments, like the height, vertical offset and the number of lines, but the current implementation proved to be the most flexible. One should be aware of the fact that the offsets depend on the design of the glyphs used.

4

```
\def\DroppedCaps#1#2#3#4#5#6#7%
  {\par
   \vskip#6\baselineskip
   \penalty-200
   \vskip-#6\baselineskip
   \setbox0=\hbox
     {\font\temp=#2 at #3%
      \temp#1{#7}\hskip#4}%
   \setbox0=\hbox
     {\lower#5\box0}%
   \ht0=\ht\strutbox
   \dp0=\dp\strutbox
   \hangindent\wd0
   \hangafter-#6%
   \noindent
   \hskip-\wd0
   \vbox{\forgetall\box0}%
   \nobreak}
```

Before we go to the next topic, we summarize this command:

```
\DroppedCaps
   {command} {font}
   {height} {hoffset} {voffset} {lines}
```

\TreatFirstLine

INSTEAD OF LIMITING ITS ACTION TO ONE TOKEN, THE NEXT MACRO TREATS THE whole first line. This paragraph was typeset by saying:

```
\TreatFirstLine {\sc} {} {} {}
Instead of limiting its action to one token, the next macro
treats the whole first line. This paragraph was typeset by
saying:
```

The combined color and font effect is also possible, although one must be careful in using macros that accumulate grouping, but the commands used here are pretty save in that respect.

```
\TreatFirstLine {\startcolor[red]\bf} {\stopcolor} {} {}
The combined color and font effect is also possible,
```

```
although one must be careful in using macros that accumulate
grouping, but the commands used here are pretty save in that
respect.
```

Before we explain the third and fourth argument, we show the implementation. Those who know a bit about the way TEX treats tokens, will probably see in one glance that this alternative works all right for most text–only situations in which there is enough text available for the first line, but that more complicated things will blow. One has to live with that. A workaround is rather trivial but obscures the principles used.

```
\def\TreatFirstLine#1#2#3#4% before, after, first, next
  {\leavevmode
   \bgroup
   \forgetall
   \bgroup
   #1%
   \setbox0=\box\voidb@x
   \setbox2=\box\voidb@x
   \def\grabfirstline##1 %
     {\setbox2=\hbox
        {\ifvoid0
           {#3{\ignorespaces##1}}%
         \else
           \unhcopy0\ {#4{##1}}%
         \fi}%
      \ifdim\wd2=\!!zeropoint
        \setbox0=\box\voidb@x
        \setbox2=\box\voidb@x
        \let\next=\grabfirstline
      \else\ifdim\wd2>\hsize
        \hbox to \hsize{\strut\unhbox0}#2\egroup
        \break##1\
        \egroup
        \let\next=\relax
      \else
        \setbox0=\box2
        \let\next=\grabfirstline
      \fi\fi
      \next}%
   \grabfirstline}
```

individual words. Of course one needs ... to know a bit more about the macro package used to get real nice effects, but this example probably demonstrates the principles well.

```
\gdef\FunnyCommand
  {\getrandomfloat\FunnyR{0}{1}%
   \getrandomfloat\FunnyG{0}{1}%
   \getrandomfloat\FunnyB{0}{1}%
   \definecolor[FunnyColor][r=\FunnyR,g=\FunnyG,b=\FunnyB]%
   \color[FunnyColor]}

\TreatFirstLine {\bf} {} {\FunnyCommand} {\FunnyCommand}
The third and fourth argument can be used to gain special
effects on the individual words. Of course one needs ...
```

Like in dropped caps case, one can hide such treatments in a macro, like:

```
\def\MyTreatFirstLine%
  {\TreatFirstLine{\bf}{}{\FunnyCommand}{\FunnyCommand}}
```

```
\beginofshapebox
When using \CONTEXT, one can also apply this funny command
to whole lines by using the reshape mechanism. Describing
this interesting mechanism falls outside the scope of this
module, so we only show the trick. This is an example of
low level \CONTEXT\ functionality: it's all there, and it's
stable, but not entirely meant for novice users.
\endofshapebox

\reshapebox{\FunnyCommand{\box\shapebox}} \flushshapebox
```

This mechanism permits hyphenation and therefore gives better results than the previously discussed macro `\TreatFirstLine`.

`\TreatFirstCharacter`

Just to be complete we also offer a very simple one character alternative, that is not that hard to understand:

6

```
\def\TreatFirstCharacter#1#2% command, character
  {{#1{#2}}}
```

A previous paragraph started with:

```
\TreatFirstCharacter{\bf\color[green]} Just to be
```

`\StackCharacters`

The next hack deals with vertical stacking.

7

```
\def\StackCharacters#1#2#3#4% sequence vsize vskip command
  {\vbox #2
     {\forgetall
      \baselineskip0pt
      \def\StackCharacter##1{#4{##1}\cr\noalign{#3}}%
      \halign
        {\hss##\hss&##\cr
         \handletokens#1\with\StackCharacter\cr}}}
```

Such a stack looks like:

and is typeset by saying:

```
\StackCharacters{CONTEXT}{}{\vskip.2ex}{\FunnyCommand}
```

An alternative would have been

```
\StackCharacters {CONTEXT} {to 5cm} {\vfill} {\FunnyCommand}
```

At a lower level horizontal and vertical manipulations are already supported by: \processtokens

```
\processtokens {begin} {between} {end} {space} {text}
```

This macro is able to typeset:

L E T ' S H A V E F U N

which was specified as:

```
\processtokens
  {\hbox to .5\hsize\bgroup} {\hfill}
  {\egroup} {\space} {LET'S HAVE}

\processtokens
  {\vbox\bgroup\raggedcenter\hsize1em}
  {\vskip.25ex} {\egroup} {\strut} {FUN}
```

Next we introduce some font manipulation macros. When we want to typeset some text spread in a well defined area, it can be considered bad practice to manipulate character and word spacing. In such situations the next few macros can be of help: \NormalizeFontHeight \NormalizeFontWidth

```
\NormalizeFontHeight \name {sample text} {height} {font}
\NormalizeFontWidth  \name {sample text} {width}  {font}
```

These are implemented using an auxilliary macro:

```
8  \def\NormalizeFontHeight%
     {\NormalizeFontSize\ht}
9  \def\NormalizeFontWidth%
     {\NormalizeFontSize\wd}
10 \def\NormalizeFontSize#1#2#3#4#5%
     {\setbox0=\hbox{\font\temp=#5 at 10pt\temp#3}%
      \dimen0=#10
      \dimen2=10000pt
      \divide\dimen2 by \dimen0
      \dimen4=#4%
      \divide\dimen4 by 1000
      \dimen4=\number\dimen2\dimen4
      \edef\NormalizedFontSize{\the\dimen4}%
      \font#2=#5 at \NormalizedFontSize}
```

Consider for instance:

```
\NormalizeFontHeight \tmp {X} {2\baselineskip} {cmr10}
{\tmp To Be Or Not To Be}
```

This shows up as (we also show the baselines):

1 To Be Or Not To Be

The horizontal counterpart is:

```
\NormalizeFontWidth \tmp {This Line Fits} {\hsize} {cmr10}
\hbox{\tmp This Line Fits}
```

The calculated font scale is avaliable in the macro \NormalizedFontSize.

One can of course combine these macros with the ones described earlier, like in:

```
\NormalizeFontHeight \DroppedFont {2\baselineskip} {cmbx12}

\def\NicelyDroppedCaps%
  {\DroppedCaps
     {\kleur[groen]}
     {\DroppedFont}
     {2pt}
     {\baselineskip}
     {2}}
```

It's up to the reader to test this one.

```
\protect
\endinput
```