Some funny macro’s

Keywords
dropped caps, CONThXT

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.

\DroppedCaps
\unprotect

This module implements some typographics tricks that can be fun when designing document layouts. The examples use macros that are typical to CONTXThXT, but non CONThXT 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.

\input supp-mis.tex
\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
\unprotect

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
\unprotect

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
\unprotect
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 guarantee consistency throughout the document:

\begin{verbatim}
\def\MyDroppedCaps{
\DroppedCaps{\color[green]}{cmbx12}{3\baselineskip}{3pt}{3\baselineskip}{4}}
\MyDroppedCaps
\end{verbatim}

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.

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

\begin{verbatim}
\DroppedCaps{command}{font}{height}{hoffset}{voffset}{lines}
\end{verbatim}

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

\begin{verbatim}
\TreatFirstLine{\startcolor[red]\bf}{\stopcolor}
\end{verbatim}

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.

\begin{verbatim}
\TreatFirstLine{\startcolor[red]\bf}{\stopcolor}
\end{verbatim}

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.

\begin{verbatim}
\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\unhcopy0}#2\egroup
      \break##1\ 
      \egroup
    \let\next=\relax
    \else
      \setbox0=\box2
      \let\next=\grabfirstline
    \fi
  \fi
  \next}
\grabfirstline}
\end{verbatim}

The third and fourth argument can be used to gain special effects on the 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.

\begin{verbatim}
\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
\end{verbatim}

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

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

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

A previous paragraph started with:

The next hack deals with vertical stacking.

Such a stack looks like:

and is typeset by saying:

An alternative would have been
At a lower level horizontal and vertical manipulations are already supported by:

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

This macro is able to typeset:

\setbox0 = \hbox {\strut LET'S HAVE FUN}

which was specified as:

\processtokens \hbox {\hfill LET'S HAVE}
\processtokens \vbox \raggedcenter \hsize1em \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 \name {sample text} {height} {font}
\NormalizeFontWidth \name {sample text} {width} {font}

These are implemented using an auxiliary macro:

8 \def \NormalizeFontHeight\ht {
9 \NormalizeFontSize \ht
10 \def \NormalizeFontWidth\wd {

Consider for instance:

\NormalizeFontHeight \tmp {X} {2\baselineskip} {cmr10}
\NormalizeFontWidth \tmp {This Line Fits} {hsize} {cmr10}
\hbox {\tmp This Line Fits}
The calculated font scale is available in the macro \NormalizedFontSize.

This Line Fits

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

\NormalizeFontHeight \DroppedFont \baselineskip \texttt{\texttt{cmbx12}}
\def\NicelyDroppedCaps{
\DroppedCaps
{\color\groen}
{\DroppedFont}
{2pt}
{\baselineskip}
{2}}

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

11 \protect
12 \endinput