The Document Style Designer as a Separate Entity

Victor Eijkhout
University of Illinois, CSRD, 305 Talbot Lab., 104 S. Wright St., Urbana, IL 61801
eijkhout@s12.csrd.uiuc.edu

Andries Lenstra
Department of Mathematics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands
lenstr@sci.kun.nl

Abstract

An argument for the need for a programmable meta format: a
format that introduces a new syntactic level in TeX for document
style designers.

TeX has a number of characteristics that set
it apart from all other text processors. Its unsur-
passed quality of text setting and its capabilities for
handling mathematics are some of the more visible
aspects. On a deeper level, however, the extreme
programmability of TeX is just as big an asset. Any
layout can be automated to an arbitrary extent. (It
is strange that *The TeXbook* gives almost no hint of
this.)

The form such automation usually takes is
what is called 'generalized markup'. The person
who keys in the text has at his or her disposal
commands that describe the logical structure of the
document, and as little as is possible of the visual
structure.

Document styles as they appear in LaTeX are
examples of this. Here the layout is not merely
automated, it is completely hidden from the end
user. In particular, the same input can produce
widely different output by letting it be interpreted
by different styles.

With LaTeX, however, the problem is the
production of document styles. This is a major
task, and consequently most people use either the
standard styles, or minor modifications of these.
Furthermore, LaTeX does not offer sufficient tools
to produce even moderate variations on the layout
of the standard styles.

For scientific use of TeX one can become
reconciled to this situation. A scientist should be
more concerned with the contents than with the
looks of a document, so if there is a format that
offers all the tools to get those contents across to
the reader, the visual appearance is of secondary
concern. We may conclude that, for scientists,
LaTeX fits the bill.

When a document is not a scientific article,
however, the inflexibility of LaTeX renders it useless.
The alternative would seem to be plain TeX, but
there the objection is the long and slow learning
curve.

One way out of this dilemma is the 'front
end to TeX' approach taken in *The Publisher* from
Arbortext and *Grif* from Gipsi. Both systems
present almost a 'wysiwyg' (what you see is what
you get) interface to the user (the term 'wysipn',
what you see is pretty neat, has been used), and
allow altering style parameters via dialog boxes and
menus. In both cases, however, programming the
basic style structure and appearance is still less
than trivial.

In this article we describe an approach which
brings down the complexity of programming a style
to that of using it. We propose a front end
programming language for style design, which is
itself implemented in TeX.

The 'Checklist' Approach to Style Definition

If one compares TeX to mouse-and-menu text pro-
cessor packages, one runs into two basic characteris-
tics of programming that constitute a disadvantage
when learning TeX.

The obvious first point is that TeX has a
syntax. Every TeX programmer knows that you
can have no end of fun with missing or mismatched
braces. Mouse riders are not bothered by this. One
can, at most, click the wrong item, but one cannot
click in the wrong way.

The second point is more subtle: programming
involves and imposes algorithmic thinking. Consider
the ordinary loop statement

```
for $i$:=$1$ to $n$
```

In many cases, all that is meant is

```
for all $i$ between $1$ and
         $n$ inclusive
```

Thus, the syntactic formulation contains a sequentiality that may semantically not be present. Similarly, the statements in a macro definition are sequentially ordered, even when the corresponding actions are in no such relation.

But even when actions are sequentially ordered, it should not always be necessary for a style designer to specify them in that same order. For instance, in the design of a list environment the amounts of white space above and below the list, and the amount of indentation of the list, should be specifiable in any order, even though they correspond to sequentially ordered actions. Also, the decision whether or not to break a page above a certain type of heading should be specifiable at any point in the definition.

Such actions do not really correspond to design decisions, rather they are the specific incarnations of general parameters and switches. Thus, it would be a valid approach to style design to offer the person implementing the style a small number of basic constructs (these could for instance be headings, lists, and page layouts), each of which has a 'checklist' of parameters and switches controlling the final layout.

Abandoning sequentiality, and making most specifications optional with default values, is then a sensible way to facilitate TeX programming. An implementation of these demands could take the form of lists of keyword-value pairs. Such lists can be presented in extremely simple syntax, and as matching is performed by keyword instead of by position, such an approach would meet some of the criticism of algorithmic specification above.

As an example, the layout of unnumbered section headings could be given in Lisp 'property list' syntax as,

```
\defineheading:section
    (white_above 6pt plus 2pt minus 1pt)
    (white_below 6pt plus 2pt minus 1pt)
    (caption (size 12pt) (style bold)
             (shape ragged_right))
```

but any other syntax is just as valid. It is advisable, however, to steer clear of the idiosyncrasies of the pure TeX syntax.

Metaformats

Checklists may capture a large part of the variation in basic constructs, but for any set of parameters

there will be a layout that eludes classification in terms of these parameters. Thus, it seems inevitable that a style implementer needs to do some programming. However, it is possible to make a very smooth transition between marking a checklist and programming macros. For this, we need the distinction between formats and 'metaformats'.

Let us denote by the term 'format' any collection of macros that gives the end user commands of a higher level than those of pure TeX. By 'metaformat,' we will mean a format such that the commands for the end user are in majority not defined in the format. Metaformats offer the tools with which a style implementer constructs the commands for the end user.

In a restricted sense, the LaTeX format is a metaformat. A good example here is the \@startsection macro, which is basic to LaTeX, and in terms of which commands like \section are defined in the style files. The parameters of this command are mainly numeric parameters determining the layout. One parameter functions as a switch.

As another example, the \list command, which is the basis for various environments in LaTeX, offers the style designer the possibility for programmable extension. Such extensions are specified by passing a piece of TeX code as the second parameter, that is, LaTeX does not really provide the style implementer with a separate syntactic level.

Calling LaTeX a 'parameterized metaformat,' we can also envision 'programmable metaformats.' There is no objection against implementing a new programming language in TeX which would be easier to learn and to use.

The challenge is then to find primitives that allow formulation in a simple syntax and that are sufficiently powerful for producing a wide range of layouts.

By any other name ...

One choice for the primitives of a metaformat is immediately clear: the boxes and glue of TeX itself. Any TeX programmer knows that you can do all typesetting with boxes and glue, so why not give them to a style designer? The problem of course is how to simplify their declaration. It is here that the writer of the metaformat takes certain decisions.

Consider, as an example, section headings such as they appear in the LaTeX article style. These

1. Section title
1.1. Subsection title

take the form (but not the implementation) of two

boxes on the same line: one containing the number, the other containing the text. Also the headings of the 'artikel' style can be described thus. But the

1. Section title
1.1. Subsection title

box of the number then has a prescribed width. In both cases, however, the sum of the width is the text width. We can therefore imagine that these two layouts were specified as

```
\defineheading:section
    (inline
        ((value sectionnumber)
         (spaces 2))
        (title))
```

for the standard LATEX heading, and

```
\defineheading:section
    (inline
        ((value sectionnumber)
         (FillUpTo labelwidth))
        (title))
```

Here the metacommand 'inline' is just an \hbox to \hsize in disguise. It takes all boxes and sets them at natural or specified width, and the width of the box containing the actual heading is calculated to fill the remainder of the text width.

Headlines and footlines provide a nice example of how a programmable metaformat can make intelligent use of TEX's glue. Like section headings, footlines are a disguised \hbox to \hsize. A footline with just a left aligned page number can be specified like

```
(footline (value pagenumber))
```

where the format supplies a trailing \hfil to prevent an underfull box. Cases where the number should be right aligned can be specified as

```
(footline (whitespace fillerup)
          (value pagenumber))
```

where the 'whitespace' is an \hfill, squashing the trailing \hfil at the end of the line.

A syntax and instruction set, such as sketched in these examples, tax the programming capabilities of the format designer, but not those of the style designer. In effect, the format designer implements a new programming language on top of TEX with a simple syntax, a small instruction set, and at the same time, sufficient generality to produce a wide range of layouts.

Obviously, a smaller instruction set makes learning the format easier. Another advantage is less immediately clear: it reduces the chance of errors. More compact instructions will likely have a more defined function; so on the one hand the style implementer need not specify those actions that must be taken anyway, and on the other hand the format can perform some consistency checking on the intentions of its user.

Conclusion

We have argued the need for a 'programmable metaformat:' a format that introduces a new syntactic level in TEX for document style designers. Such a level should probably have a different syntax from pure TEX, and it should contain a relatively small instruction set in which the actual user macros are written. We have indicated that elements of such a format can, to a large extent, be captured in 'checklists.' Others can take the form of intelligently disguised TEX primitives. We believe that a format incorporating these principles is possible, and that it can be taught to people with little knowledge of TEX.

As an illustration of these ideas we present the style definition of this article in the 'Lollipop' format.

Example

The following piece of code is the style definition for this article given in the 'Lollipop' format.

```
\typeface:Computer
\fontsize:10 \fontstyle:roman

%declare \parindent
\distance:indentation=20pt
%white space around text elements
\distance:whiteline=6pt plus 2pt
        minus 2pt
\distance:leadingwhite=whiteline
\distance:trailingwite=0pt

%lots of defaults used here
\defineheading:section
        fontstyle:bold stop

%very simple page layout
\definelayout:twocolumn
    height:22.5cm width:16.5cm
    band:start column
        whitespace:0.6cm
        column
        band:end
    stop
\twocolumn %install output routine

%paragraph shape
```

```
\defineparagraph:flushright indent:yes
    rightjustified:yes stop
\flushright %install paragraph shape

%bibliography
\definelist:literature counter:1
    item:left
        litteral:[ value:itemnumber
        litteral:]
    item:end stop
\externalreferences:yes
```

Bibliography

Braams, Johannes, Victor Eijkhout, and Nico Poppelier, "The development of national LaTeX styles," *TUGboat*, Vol 10(4), 1989.

Grif manual / Les languages de Gipsi, Gipsi 1989

Knuth, Donald E. *The TEX book*, Addison-Wesley Publishing Company, 1984.

Lamport, Leslie. *LaTeX, a document preparation system*, Addison-Wesley Publishing Company, 1986.