

Any changes made by `\everypar` are now effected *inside a group*. In this case one remedy is to insert a `\leavevmode` command, or to define

```
\def\smallcapswords#1{\leavevmode
  {\smallcaps #1}}
```

which can be used at any place.

Another remedy would be to let all assignments controlling indentation be global. However, there are some subtle objections to this.

3 About macro packages and users

Above I remarked that plain \TeX does not use `\everypar`, and that \LaTeX redefines it a lot. This means that in plain \TeX the user is free to take every value of `\everypar` that he or she likes; in \LaTeX every attempt of the user to use `\everypar` is immediately thwarted.

One might ask how the use of `\everypar` that I have sketched compares to this. Can the user be allowed to access `\everypar`, even if the macro package needs it all the time?

In my own 'Lollipop' format I have taken the following way out. The user or the style designer is allowed to fill in `\everypar`, as long as the statement

```
\the\everyeverypar
```

is included. Here `\everyeverypar` is the token list with the constant actions such as indentation control that should be performed always.

A format designer who wishes to hide even this from the user or the style designer, could use the following piece of code

```
\newtoks\temppar
\def\everyparagraph
  {\afterassignment\xevpar
   \temppar}
\def\xevpar
  {\edef\act{\everypar=
    {\the\everyeverypar
     \the\temppar
    }}%
   \act}
```

so that it becomes possible to write

```
\everyparagraph={\DoSomething
  \everyparagraph={}}
```

while the `\everypar` will still contain all of the constant actions.

Short explanation: `\everyparagraph` is a macro that is made to look like a token parameter by the use of `\afterassignment`. This latter command sets aside `\xevpar` for execution after whatever follows is assigned to `\temppar`. Following the assign-

ment, the macro `\xevpar` unwraps the `\temppar` token list and the constant actions into `\everypar`.

4 Conclusion

In a systematic layout indentation commands need never be typed by the user; they can all be hidden in macros. Using `\everypar` it is possible to prevent indentation both in single instances, and throughout the document. This has the advantage that it is not necessary to zero the `\parindent` parameter or use `\indent` and `\noindent` instructions.

The approach of employing `\everypar` as sketched above can also be used for a paragraph skip scheme, as I will show in the subsequent article.

References

- [1] Donald Knuth, *The \TeX book*, Addison-Wesley Publishing Company, 1984.
- [2] Leslie Lamport, *\LaTeX , a document preparation system*, Addison-Wesley Publishing Company, 1986.
- [3] Victor Eijkhout, Unusual paragraph shapes, *TUGboat* vol. 11 (1990) #1, pp. 51-53.
- [4] Stanley Morison, *First principles of typography*, Cambridge University Press, 1936.
- [5] J. Braams, V. Eijkhout, N.A.F.M. Poppelier, The development of national \LaTeX styles, *TUGboat* vol. 10 (1989) #3, pp. 401-406.

A `\parskip` Scheme

Victor Eijkhout

While I was working on the \LaTeX styles described in [1], it became apparent to me that lots of people are rather fond of the sort of layout that can be described as

```
\parindent=0cm
\parskip=6pt % or other positive size
```

Unfortunately, most of them realize this layout by no more sophisticated means than simply inserting these two lines at the beginning of the input. The drawback of such a simple action is that all sorts of vertical spaces are augmented by the `\parskip` when there is absolutely no need to, or where it is positively unwanted. Examples of this are the white space below section headings, and the white space above and below list environments in \LaTeX .

In this article I will present an approach that unifies the paragraph skip and the white spaces surrounding various environments. Since the macros given below make use of the `\everypar` token list, this article may be seen as a sequel to the previous article in this issue of *TUGboat* concerning an indentation scheme, which is based on a similar principle. The `\everypar` parameter was explained there.

1 `\parskip`

`TEX` starts a paragraph when it switches from vertical to horizontal mode. The vertical mode may have been initiated by a `\par` (for instance because of an empty line after a preceding paragraph) or by a vertical skip command; the switch to horizontal mode can be effected by, for example, a character or a horizontal skip command (see the list in [2, p. 283]). Immediately above the first line of the paragraph `TEX` will then add glue of size `\parskip` to the vertical list¹.

Apparently, then, the `\parskip` parameter is very simple to use. That this is only an apparent simplicity becomes clear in a number of instances.

For instance, unless precautions are taken, the white space below headings is augmented by the paragraph skip. Precautions against this are not particularly elegant: the easiest solution is to include a

```
\vskip-\parskip
```

statement, to backspace the paragraph skip in advance. Such an approach, however, is somewhat error-prone. Vertical spacing will be messed up if what follows is not a paragraph, but a display formula or a box.

Similar considerations apply to the amounts of white space that surround, for example, list environments, as in `LATEX`.

2 Paragraph skip: to be or not to be

(This section is something of a footnote to the rest of the article. Readers who are not interested in layout considerations may skip the rest of it.)

Ordinarily in plain `TEX` and in `LATEX` the paragraph skip is set to `0pt plus 1pt`, which gives pages some 'last resort' stretchability. However, even an amount of vertical space as small as one point may become very visible, and often without need (see for instance the first page of the preface of [2]).

Furthermore, Stanley Morison states that not indenting paragraphs is 'decidedly an abject

¹ Unless this paragraph is at the start of a vertical list, for instance at the start of a vertical box or insertion item.

method' [3]. However, reading his intention instead of his words, he is only concerned with the recognizability of the individual paragraphs. The positive value of the paragraph skip is sufficient to ensure this. If a layout is based on zero values for both `\parindent` and `\parskip`, one may for instance give the `\parfillskip` a positive natural width to prevent last lines of a paragraph from almost, or completely, lining up with the right margin.

Neither Donald Knuth nor Leslie Lamport seems to have given much thought to the case where the paragraph skip has a positive natural width. Leslie Lamport dismisses all potential difficulties with the remark that 'it is customary not to leave any extra space between paragraphs' [4, p. 94].

3 Environments and white lines

Given that the paragraph skip appears to interact with explicit vertical spacing in user macros, it may seem like a good idea to find a unified approach to both. In the rest of this article I will describe the implementation of the following basic idea: *give the paragraph skip the value zero whenever you do an explicit vertical skip.*

For the presentation I assume a context with some form of environments. These are the assumptions that I make about such environments:

- An environment is a portion of material that is vertically separated from whatever is before and after it. Thus, according to this definition, a portion of a paragraph cannot be an environment, nor can an environment start or end in the middle of a paragraph.
- An environment has associated with it three glue parameters: to an environment `foo` correspond `\fooStartskip` (glue above the environment), `\fooParskip` (the paragraph skip inside the environment), and the `\fooEndskip` (glue below the environment).
- At the outset of the environment a

```
\StartEnvironment{foo}
```

statement is executed; at the end of the environment a macro

```
\EndEnvironment{foo}
```

is executed. These statements are assumed to contain a `\begingroup` and `\endgroup` respectively.

Such assumptions are sufficiently general for the macros below to be adaptable to existing macro packages. At first sight it would appear as if section headings are not covered by the above points. However, there is no argument against the start and end of an environment occurring in the same macro.

4 Tools

First I will present two auxiliary macros: `\csarg` and `\vspace`.

The command `\csarg` is only needed inside other macros; it is meant to enable constructs such as

```
\csarg\vskip{#1Parskip}
```

Its definition is

```
\def\csarg#1#2{\expandafter
  #1\csname#2\endcsname}
```

By way of explanation of this macro, consider a simple example. Let us assume that there exists a macro

```
\def\startlist#1{ ...
  \csarg\vskip{#1Startskip}
  ...}
```

The call

```
\startlist{enumerate}
```

will then lead to the following call to `\csarg`:

```
\csarg\vskip{enumerateStartskip}
```

This expands to

```
\expandafter\vskip
  \csname enumerateStartskip\endcsname
```

Now the `\expandafter` forces `\csname` to be executed before the `\vskip`, so the next step of the expansion looks like

```
\vskip\enumerateStartskip
```

and this statement can simply be executed.

Next I need a generalization of `\vskip`, which I will call `\vspace`: a number of calls to `\vspace` should have the effect that only the maximum argument is placed.

```
\newskip\tempskipa
\def\vspace #1{%
  \tempskipa=#1\relax
  \ifvmode \ifdim\tempskipa<\lastskip
    \else \vskip-\lastskip \vskip\tempskipa
  \fi
  \else \vskip\tempskipa \fi}
```

This may need some explanation, too. First, by the assignment

```
\tempskipa=#1
```

I allow the argument of `\vspace` to be both a control sequence, for instance `\parskip`, and a denotation, for instance `5pt plus 3pt`. If one omits the assignment, the latter option would cause trouble in the `\ifdim` test.

The decision to keep the maximum value of the skip, instead of always replacing the last skip, was

motivated by phenomena such as a display formula at the end of a list. If the skip below the display is larger than the vertical glue below the list (which may for instance be zero), the former should be retained entirely.

Note that this macro will insert its argument even if it has the same size as the last skip. There is a good reason for this. If the call to `\vspace` follows a `\par` command at the end of a paragraph, it is called in vertical mode, but the last item on the vertical list is a box (the last line of the paragraph) instead of a glue item. The parameter `\lastskip` will then be zero. If the argument to `\vspace` is something like `0pt plus 5pt` we still want it to be added to the list, even though its natural size is zero.

5 The environment macros

In this section, I will give the implementation of the macros `\StartEnvironment` and `\EndEnvironment`.

There is a remarkable similarity between these two macros. As I explained above, the basic idea is to have only explicit spacing above and below the environment; thus, the value of `\parskip` should then be zero both for the first paragraph in the environment, and for the first paragraph that follows it. Both macros should then

- save the current value of the paragraph skip;
- set the paragraph skip to zero;
- give T_EX a signal that, somewhere in the near future, the old value of `\parskip` is to be restored.

For this I allocate a skip register and a conditional:

```
\newskip\TempParskip
\newif\ifParskipNeedsRestoring
```

The basic sequence for the starting and ending macros is then

```
\TempParskip=\parskip
\parskip=0cm\relax
\ParskipNeedsRestoringtrue
```

For both macros, however, this sequence needs to be refined slightly.

The paragraph skip to be 'restored' at the start of the environment is the specific value associated with that environment. This gives us:

```
\def\StartEnvironment
  #1{\csarg\vspace{#1Startskip}
  \begingroup %% make changes local
  \csarg\TempParskip{#1Parskip}
  \parskip=0cm\relax
  \ParskipNeedsRestoringtrue}
```

Note that the statement

```
\csarg\TempParskip{#1Parskip}
```