```
  \or\afterassignment\afb@xagarg
  \fi
  \setbox\afbox}
\def\afb@xagarg{\aftergroup\afb@xarg}
```

First, `\afterbox` puts the ⟨argument⟩ into `\afb@xarg`. Then the `\chardef` command reads a ⟨number⟩ which turns out to be a ⟨normal integer⟩ with a ⟨character token⟩ (see *The TEXbook*, p. 269). As the syntax of ⟨number⟩ requires, TEX expands tokens and looks for ⟨one optional space⟩ which turns out ⟨empty⟩. This looks crazy, but it has the effect of unpacking the first non-expandable token of ⟨box⟩ if it was hidden behind expandable tokens like `\null` or `\line` (or `\Boxit` below). This non-expandable token's meaning is then assigned to `\next` and tested by `\afb@xtest`. It must be one of the seven primitives listed with the `\ifx`s, and the cases 1 and 2 correspond to the two behaviours of `\afterassignment` mentioned above. In both cases, `\afb@xarg` will reappear exactly at the time when the `\setbox` assignment is finished, e.g.:

```
\afterbox \t \box1
```

results in `\setbox\afbox=\box1 \t`, whereas

```
\afterbox \t \hbox{h}
```

first becomes `...\hbox{\afb@xagarg h}` and then results in `\setbox\afbox=\hbox{h}\t`.

For example,

```
\def\Boxit{\hbox\bgroup\afterbox
   {\vrule
    \dimen0=\dp\afbox
    \advance\dimen0 by3.4pt
    \lower\dimen0 \vbox
     {\hrule \kern3pt
      \hbox{\kern3pt\box\afbox\kern3pt}
      \kern3pt \hrule}%
    \vrule \egroup}}
```

solves Ex. 21.3 of *The TEXbook* with `\Boxit<box>` instead of `\boxit{<box>}`, and `\Boxit<box>` is itself a ⟨box⟩, so that `\Boxit\Boxit<box>` makes a double frame. The macro `\framedhbox` defined by

```
\def\framedhbox{\Boxit\hbox}
```

can be used exactly like the primitive `\hbox`:

```
\framedhbox{<horizontal material>}
```

It can also be `\raised`, or assigned to a box register, and `to` or `spread` can be specified.

◇ Sonja Maus
Memelweg 2
5300 Bonn 1
Federal Republic of Germany

An Indentation Scheme

Victor Eijkhout

Indentation is one of the simpler things in TEX: if you leave one input line open you get a new paragraph, and it is indented unless you say `\noindent`. And if you get tired of writing `\noindent` all of the time, you declare

```
\parindent=0pt
```

at the start of your document. Easy.

More sophisticated approaches to indentation are possible, however. In this article I will sketch a quite general approach that can easily be incorporated in existing macro packages. For a better appreciation of what goes on, I will start with a tutorial section on what happens when TEX starts a paragraph.

1 Tutorial: paragraph start

When TEX is not busy typesetting mathematics, it is processing in *horizontal mode*, or *vertical mode*. In horizontal mode it is putting objects — usually characters — next to each other; in vertical mode it is putting objects — usually lines of text — on top of each other.

To see that there is a difference, run the following pieces of code through TEX:

```
\hbox{a}
\hbox{b}
\bye
```

and

```
a
\hbox{b}
\hbox{c}
\bye
```

You notice that the same objects are treated in two different ways. The reason for this is that TEX starts each job in vertical mode, that is, stacking material. In the second piece of input TEX saw the character 'a' before it saw the boxes. A character is for TEX the sign to switch to horizontal mode, that is, lining up material, and start building a paragraph.

Commands that can make TEX switch to horizontal mode are called 'horizontal commands'. As appeared from the above two examples characters are horizontal commands, but boxes are not. Let us now look at the two most obvious horizontal commands: `\indent` and `\noindent`.

1.1 `\indent` and `\noindent`

`\indent` is the command to start a paragraph with indentation. TEX realizes the indentation by insert-

ing a box of width \parindent. If you say \indent somewhere in the middle of a paragraph you get some white space there, caused by the empty box.

\noindent is the command to start a paragraph without indentation. After this command TEX merely switches to horizontal mode; no indentation box is inserted. If you give this command somewhere in the middle of a paragraph it has no effect at all.

If TEX sees a horizontal command that is not \indent or \noindent, for instance a character, it acts as if the command was preceded by \indent. This is why paragraphs usually start with an indentation.

As an illustration here is a small variation on the above two examples:

```
\noindent
\hbox{a}
\hbox{b}

\indent
\hbox{a}
\hbox{b}
\bye
```

Now in both cases the boxes are part of a paragraph that was explicitly begun with \indent or \noindent.

1.2 Using \everypar

TEX performs another action when it starts a paragraph: it inserts whatever is currently the contents of the token list \everypar. Usually you don't notice this, because the token list is empty in plain TEX (the TEX book [1] gives only a simple example, and the exhortation 'if you let your imagination run you will think of better applications'). LATEX [2], however, makes regular use of \everypar. Some mega-trickery with \everypar can be found in [3].

- Just to show how this works, I put in front of this paragraph the statement

```
\everypar={$\bullet\quad$}
```

That is, I told TEX that $\bullet\quad$ should be inserted in front of a paragraph.

- There's nothing specified for this paragraph; I get the bullet for free, as \everypar does exactly what its name promises: it is inserted in front of *every* paragraph.

At the end of the previous paragraph I specified

```
\everypar={}
```

so nothing is inserted from this paragraph onwards.

1.3 Removing indentation

Every TEX user knows that indentation can be prevented globally by setting \parindent to zero. However, this is rather crude, and if you use the plain TEX macros you may notice several rather unpleasant side effects of this action, for instance when you use the macros \item and \footnote.

It is possible to use \everypar to prevent indentation, or more correctly: to remove indentation. This can be achieved by

```
\everypar={{\setbox0=\lastbox}}
```

This needs some explanation.

If the last item that was processed by TEX is a box, then that box is accessible by the command \lastbox. If the last item was not a box then \lastbox is an empty box, but no error ensues. As the \everypar list is inserted after any indentation box, the \lastbox command will get hold of the indentation box if there is one. By assigning the last box to another box register — here \box0 — it is removed from where it was previously.

Finally, the statement

```
\setbox0=\lastbox
```

is enclosed in braces. TEX's grouping mechanism restores values when the group ends that were current when the group began. In this case it has the effect of totally removing the indentation box: first it is taken and assigned to \box0, then the value of \box0 is restored to whatever it was before the group began.

1.4 Other actions at the start of a paragraph

In the above discussion I have omitted one action that takes place at the start of a paragraph: TEX inserts (vertical) \parskip glue above the paragraph. As this has no relevance for the subject of indentation I will not go into it any further. However, in a subsequent article I will give more information about \parskip.

2 To indent or not to indent

In classical book typography [4] every paragraph is indented, with the exception of the first paragraph of a chapter or section. Nowadays a design where no paragraph indents is quite common. There are two mixtures between always indenting and never indenting: occasionally indenting, and occasionally not indenting. Thus it seems possible to characterize indentation strategies by two yes/no parameters: one that decides whether paragraphs should indent in principle, and another parameter that can over-

rule those decisions. Let us now see how this can be implemented in TEX.

2.1 Implementation

Above I have already indicated that changes to \parindent should be avoided. Let us then assume that \parindent is greater than zero, even if we will never indent a paragraph (see [5] for other uses for the \parindent quantity). We must then realize unindented paragraphs by removing their indentation as explained above.

First we need a macro for removing the indentation:

```
\def\removeindentation
    {{\setbox0=\lastbox}}
```

Then we need the switches that control indentation:

```
\newif\ifNeedIndent %as a rule
\newif\ifneedindent %special cases
```

Now for the definition of \everypar. This is a bit tricky.

Let us first collect some bits and pieces. The main question is to decide when \removeindent should be called. This is for instance the case if \NeedIndentfalse, and that parameter is not overruled by \needindenttrue.

```
\ifNeedIndent
    \ifneedindent
    \else \removeindentation
\fi \fi
```

Indentation should also be removed when \needindentfalse overrules the general parameter \NeedIndenttrue.

```
\ifNeedIndent
\else \ifneedindent
      \else \removeindentation
\fi   \fi
```

Next we should make sure that \ifneedindent is used only for exceptional cases: if the user or a macro sets this parameter to a different value from \ifNeedIndent, then that should be obeyed exactly once.

```
\ifNeedIndent
      \ifneedindent
      \else \needindenttrue \fi
\else \ifneedindent \needindentfalse
\fi   \fi
```

This is then the full definition of \everypar:

```
\everypar={\controlledindentation}
\def\controlledindentation
   {\ifNeedIndent
          \ifneedindent
          \else \removeindentation
                \needindenttrue
          \fi
    \else \ifneedindent
                \needindentfalse
          \else \removeindentation
    \fi   \fi}
```

Another implementation would be possible:

```
\def\controlledindentation
   {\ifneedindent
    \else \removeindentation \fi
    \let\ifneedindent=\ifNeedIndent}
```

This saves one conditional, but for most paragraphs it involves an unnecessary \let command.

2.2 Usage

My aim in developing this indentation scheme was to hide all commands pertaining to indentation in macros. The user should have to specify only once whether paragraphs should indent as a rule:

```
\NeedIndenttrue
```

and then macros should declare the exceptions:

```
\def\section#1{...
    \needindentfalse
    ...}
```

2.3 But couldn't you simply ...?

Maybe people who read this have written macros themselves that end like

```
\def\section#1{...
    ...
    \noindent}
```

or

```
\def\section#1\par{...
    ...
    \noindent}
```

This works reasonably well, but it is not completely safe. In the first case there shouldn't be an empty line after a

```
\section{...}
```

call, and in the second case there can only be one empty line after

```
\section ...
```

The reason for this is that *every* empty line generates a \par command, which annuls the effect of the \noindent. Hence the more drastic approach.

An argument the other way around can also be found, by the way. As Ron Whitney pointed out to me, the following piece of code causes trouble:

```
\section{Title}
{\smallcaps The first} words are ...
```

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 paramter by the use of \afterassignment. This latter command sets aside \xevpar for execution after whatever follows is assigned to \temppar. Following the assignment, 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 is 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 TEXbook, 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.