`true` and the meaning of `\value` is defined as the attribute value.

Notice the macro `\value`: When it is passed as an argument to `\compare@with@attribute` it is still undefined. In other words, we have the funny case of a macro which — to some extent — defines the arguments, that it receives, itself.

The two examples above show rather simple applications of keyword parameters without great practical value. They should primarily be regarded as an explanation of the basic ideas how such macros can be written. In practice further extensions may be necessary. One extension may be the mixture of positional and keyword parameters, another one the definition of macros, where the keywords in the argument list may have to be reordered before they get interpreted.

The discussion on positional versus keyword parameters has a long tradition in computer science and common understanding is probably, that keyword parameters are preferable to positional ones in many cases. Also several document processing systems, e. g. Reid's *SCRIBE* system (B. K. Reid: *SCRIBE — Introductory User's Manual*, Unilogic Ltd., Pittsburgh, 1980), make use of keyword parameters to some extent. (There are even a few features in LaTeX which look like keyword parameters though Lamport does not use this terminology. See, for example, the *options* that can be given with a `\documentstyle` command.)

Using the concept of keywords parameters can probably lead to macro packages with user interfaces, which look quite different from existing ones and might be preferred by many users. Maybe even the writing of "bridgeware" macro packages to other formatting languages, for example a macro package that makes (at least certain classes of) *SCRIBE* documents processable by TeX, might become easier.

When I first thought about keyword parameters I was surprised, that it took only a few hours to write down some macros that solved the problem. So, if after all the examples above may show nothing, they at least prove once again the flexibility and power of TeX's macro language.

`\expandafter` vs. `\let` and `\def` in Conditionals and a Generalization of PLAIN's `\loop`

Alois Kabelschacht
Max-Planck-Institut für Physik

Conditionals with `\expandafter`

Sometimes the replacement text for a TeX macro should end with one or another macro call, depending on a condition. The trivial solution

```
... \if... ... \aa \else ... \bb \fi
```

works only if neither `\aa` nor `\bb` needs an argument. Otherwise a more complicated construction such as the following example from `plain.tex` is needed:

```
\def\ph@nt{\ifmmode
    \def\next{\mathpalette\mathph@nt}%
  \else\let\next\makeph@nt\fi\next}
```

There is the alternative:

```
\def\ph@nt{\ifmmode
    \expandafter\mathpalette
    \expandafter\mathph@nt
  \else\expandafter\makeph@nt\fi}
```

which uses the fact that the expansion of both `\else ... \fi` and `\fi` is empty. This alternative is definitely shorter (by 4 tokens) and as far as I can see not slower. It has the further advantage that it also works if expandable tokens are expanded but no commands are digested (e.g. in the replacement text for `\edef`). The alternative construction is clearly even more economical in such cases where one of the branches would otherwise contain a '`\let\next\relax`'.

A generalization of PLAIN's `\loop` macro

Using the above idea one could e.g. replace PLAIN's definition of `\iterate` (used in conjunction with `\loop`):

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body \let\next\iterate
    \else\let\next\relax\fi \next}
\let\repeat=\fi % this makes
      % \loop...\if...\repeat skippable
```

by

```
\def\iterate{\body
    \expandafter\iterate\else\fi}
```

Finally, omitting the `\else` and rearranging things a bit one obtains

```
\def\loop#1\repeat{\def\iterate
      {#1\expandafter\iterate\fi}%
  \iterate \let\iterate\relax}
```

which allows constructions such as

```
    \loop ... \if... ... \else ... \repeat
    \loop ... \ifcase ... \or ...
                   \else ... \repeat
```

The final '\let\iterate\relax' throws away the token list for the body of the loop which could be quite long.

TEX in the Commercial Environment Setting Multi-Column Output

Elizabeth M. Barnhart

A little more than two years ago, **TV GUIDE** magazine started to investigate the possibility of using the TEX typesetting language to compose both the national feature and local program-listing sections of the magazine. The idea of vendor independence was one of the most attractive attributes of using this as our composition language.

Academic vs. the Commercial Environment

As we started to get more involved, we discovered that a large percentage of the TEX community consisted of academic users of TEX in colleges and universities around the country, but that few commercial typesetting applications were using TEX.

The academic user is usually involved with a relatively small quantity of output — from a few pages to perhaps several hundred pages. In contrast, **TV GUIDE** publishes over 100 editions in the United States and Canada for each weekly issue. The output comes to approximately 15,000 pages per week, presenting quite a different processing problem.

In the typical academic environment, one person might key in text through a word processor or PC editor and handle the style and output of the text by the inserting of typesetting commands directly into the text. In our environment, the same keystrokes are captured once, and may repeat in several areas and in many editions of the magazine. No one single person enters the text that makes up a page of the magazine. Editors for each local station gather the programming information and send it to the main office in Radnor, Pa. Output is handled by feeding items through pre-defined typesetting-specification files.

Specific Problems

Although TEX has many positive features, we have encountered some problems as we experiment with a variety of the type elements that compose **TV GUIDE**.

One problem is that TEX was designed for much wider columns than the ones called for by our typesetting specifications. We have been able to get around this with adjustments of the `\tolerance` and penalties that control the line breaking algorithm in TEX. It would be infeasible to use the TEX defaults for these penalties, which would require frequent interacting with the copy to eliminate the many "*overfull boxes*" that would occur.

Another problem is that TEX is a paragraph setting composition language; all other composition languages that our staff had been exposed to set type line by line. In line-by-line systems, once a line of type has been hyphenated and justified, it is closed and will not be changed; TEX can rework a paragraph completely differently when one word is eliminated. This has been presenting some problems in our environment, since knowing exactly where a line breaks is important to us. Our text often includes optional copy, and we need to know how lines will fit together if optional copy in the center of a paragraph is eliminated. Taking text measurements from the longest version of the copy has been our solution to this problem.

We have also encountered difficulties with the fact that when TEX produces a `.dvi` file, the fonts involved lose their identity. They are assigned a number in the font table contained in the "postamble" of the `.dvi` file. We need to be able to convert the text back to the original format, so we must be able to reconstruct the font calls made in the original text. We are experimenting with dealing with this problem by forcing system-specific font calls into the `.dvi` file using the `\special` command.

Another one of the big problems we have encountered is the complexity of defining page layouts with "output" routines. Each section of **TV GUIDE** is different and within those sections each page can be different. One example would be switching from a three-column to a two-column format within an article. Another layout requirement is leaving drops for photographs or artwork that occupy portions of more than one column of type. We are experimenting using `\parshape` commands within output routines to deal with this problem. We may find ourselves developing a front-end page