

## Stephen Hicks

inlinedef: a general recursive token scanner with callbacks

There have been several discussions about uses of `\expandafter` that border on the ridiculous, with as many as fifteen in a row found in actual  $\TeX$  input files! Additionally, trying to expand past macro parameters `#1` causes problems because there is no guarantee that `#1` is a single token. It would instead be nice to insert something right before a single token we want to expand far in advance. A slightly more general problem is to scan tokens in the input stream while preserving spaces and grouping.

```
\let\xa\expandafter

\def\scan{\futurelet\foo\switch}
\def\switch{%
  \let\next\normal
  \ifcat\noexpand\foo\space \let\next\dospace\fi
  \ifcat\noexpand\foo\bgroup \let\next\trygroup\fi
  \ifcat\noexpand\foo\relax \try{&\meaning\foo}\fi
  \next}
\def\try#1{\ifcsname #1\endcsname\xa\let\xa\next\csname #1\endcsname\fi}
\def\dospace{\toks0\xa{\the\toks\xa0 \space}\xa\scan\unspace}
\xa\def\xa\unspace\space{ }
\long\def\trygroup#1#2#3{%
  \def\temp{#1}\xa\let\xa\next\ifx\temp\empty\recurse\else\normal\fi\next#1}
\long\def\recurse#1#2#3{%
  \begingroup\toks0{\scan#1\END{}}\xa\endgroup\xa
  \toks\xa0\xa\xa{\xa\the\xa\toks\xa0\xa{\the\toks0}}\scan}
\long\def\normal#1{\toks0\xa{\the\toks0 #1}\scan}

\def\callback#1#2#3{\def#1{\noexpand#1}\xa\def\csname&\meaning#1\endcsname#2}
```

We can set up a few callbacks, e.g. `\END` to end scanning, and `\EXPAND` to expand the next token:

```
\callback\END#1{ }
\callback\EXPAND#1{\expandafter\scan}
```

And now we can get arbitrary tokens from the input stream into `\toks0` using

```
\def\baz{!}
\scan foo {bar \EXPAND\baz} \baz \END
\message{\the\toks0} % foo\space {bar\space !}\space \baz
```

This can be made more general in several ways: if we don't check `\ifcat\noexpand\foo\relax` then we can execute callbacks on arbitrary tokens, including spaces and grouping symbols. Of course this slows things down quite a bit further, which brings me to the main disadvantage with this approach: it takes about 25 times as long than a simple string of `\expandafter`'s, and is therefore not suitable for inner loops. But the code it allows us to write, as long as efficiency isn't important, is much more readable.