Sanitizing Control Sequences Under \write

Ron Whitney

TEX's \write primitive is typically used to send information to other files for typesetting later. The most obvious examples here are those cases where chapter heads, section heads, and keywords are written to files along with corresponding page numbers to make tables of contents and indexes. (See Salomon in *TUGboat* 10, no. 3; see also Durst in the same issue for another use of \write)

The syntax of a \write statement is

\write⟨*number*⟩{⟨*token string*⟩}

where ⟨*number*⟩ is a "stream number" usually allocated by the \newwrite macro and corresponds to a disk file opened by \openout. To make matters concrete here, we will simply assume that the user has called \newwrite\outfile and that our syntax is

\write\outfile{⟨*token string*⟩}

The execution of the \write statement then either involves writing the ⟨*token string*⟩ directly (i.e. \immediately) to the file corresponding to \outfile, or writing the ⟨*token string*⟩ to a node placed in TEX's main vertical list. In the latter case, the ⟨*token list*⟩ in the node is later written to the file corresponding to \outfile as the page on which the node is placed undergoes \shipout to the dvi file.

In either case, as the ⟨*token list*⟩ is transferred to the file \outfile, it is fully expanded (that is, the ⟨*token list*⟩ is expanded as, say, it would be under an \edef; unexpandable control sequences are written out using the \escapechar and the letters or symbols which go to make up their names). There's Good News and Bad News in this. The Good News is that the information to be written to external files is often 'contained' within control sequences, and we definitely want tokens such as \number to be expanded (otherwise an index might show every item appearing on, literally, \number\pageno). The Bad News is that we would really much prefer that indexes contain items like \cos rather than their expansions (e.g. \mathop{\rm cos}\nolimits). The Bad News isn't really all that Bad, though, since we can use TEX's \noexpand and say

\write\outfile{\noexpand\cos}

to achieve what we want.

Then: Where's the Beef? The News isn't really Totally Good because our solution requires some knowledge of the contexts in which \noexpand is appropriate. \write statements are typically hidden a few levels down inside macros and one might ever know (barring authorship or clear documentation of the underlying macros) when silliness such as \noexpand is required. Lamport has done an admirable job in schooling LATEX users to use \protect for this very purpose, but even so, questions such as Chris Hand's (*TUGboat* 11, no. 3, p. 456) are natural. In Chris' case, a guillement (\<<) caused a section head to expand to some 509 characters in an .aux file, and that line was too long for TEX to handle when the .aux file was reread. It is also common to see control sequences for accents make for inscrutable tables of contents files.

A Better Method

Much nicer than user-keyboarded \noexpands would be some method of preventing expansion within macros themselves which use \write, thus not placing a user under the strain of being on the lookout for expanding arcana. This note proposes a way to handle things generally. It was suggested by a technique used by Michael Wichura in *TUGboat* 11, no. 1 along with simultaneous consideration of Peter Breitenlohner's piece on avoiding long records in \write streams in the same issue of *TUGboat*. Other people have undoubtedly thought of the same thing (see *The TEXbook*, p. 382).

The TEX primitive \meaning disgorges a "meaning" of whatever token follows it. In the case of a defined control sequence (and let's assume this control sequence has no parameters), say \foo, \meaning\foo will cause TEX to spit up the sequence macro:-> followed by a sequence of character tokens as would be obtained by \stringing the tokens of \foo's definition. The definition is thus shown as a string of character tokens, all of category 12 (except spaces).

For example, if \foo is defined to be the token string $\sin$ and $\cos$, its definition consists of 11 tokens altogether: 4 math shifts, 2 spaces, 3 letters, and 2 defined (in plain) control sequences. On the other hand, \meaning\foo produces the string

macro:->$\sin $ and $\cos $

whose right part (beyond the :->) consists of 4 space tokens and 15 character tokens of category 12. None of this material is expandable; internal objects which had been single tokens (such as \sin) are now divided into character tokens (such as \-s-i-n).

denotes an assignment (without an optional equals sign) here.

At the end of the environment we have to be more careful. It may be the case that the environment being ended was inside another environment, and occurred before the first paragraph inside that environment. In that case the value of \parskip is zero, and the proper value must still be restored. Therefore, no further actions are required. We arrive at the following implementation:

```
\def\EndEnvironment
 #1{\csarg\vspace{#1Endskip}
    \endgroup %% restore global values
    \ifParskipNeedsRestoring
    \else \TempParskip=\parskip
          \parskip=0cm\relax
          \ParskipNeedsRestoringtrue
    \fi}
```

Note that both macros start with a vertical skip. This prevents the \begingroup and \endgroup statements from occurring in a paragraph. On a side note: since these macros are executed in vertical mode, I have not bothered to terminate any lines with comment signs. Any spaces generated by these macros are ignored in vertical mode.

6 Paragraph skip restoring

So far, I have ignored one important question: how exactly is restoring the paragraph skip implemented. For this I use the \everypar token list. Basically then, the idea is the same as in "An Indentation Scheme" (p. 612): the occurrence of a paragraph will automatically have TEX perform, through the insertion of the \everypar tokens, the actions necessary for subsequent paragraphs.

```
\everypar=
   {\ControlledIndentation
      %see ''An Indentation Scheme''
    \ControlledParskip}
\def\ControlledParskip
   {\ifParskipNeedsRestoring
       \parskip=\TempParskip
       \ParskipNeedsRestoringfalse
    \fi}
```

The cost of a controlled paragraph skip is then one conditional per paragraph. Conceivably, this cost could even be reduced further (to almost zero) by defining

```
\def\CPS % Controlled Parskip
   {\ifParskipNeedsRestoring
       \parskip=\TempParskip
       \ParskipNeedsRestoringfalse
       \let\ControlledParskip=\relax
    \fi}
```

and including a statement

```
\let\ControlledParskip=\CPS
```

in both the \StartEnvironment and \EndEnvironment macros, and at the start of the job (for instance by including it in the macro package). This approach, however, does not particularly appeal to me. Too much 'pushing the bit around'.

7 Conclusion

The \parskip parameter is arguably the most tricky parameter of TEX. Its workings are very easy to describe, but in actual practice difficulties arise. In this article I have described how treatment of the paragraph skip can be integrated with the glue above and below environments. As in an earlier article on indentation, I use for this the \everypar parameter as an essential tool.

References

[1] J. Braams, V. Eijkhout, N.A.F.M. Poppelier, The development of national LATEX styles, *TUGboat* vol. 10 (1989) #3, pp. 401–406.

[2] Donald Knuth, The TEXbook, Addison-Wesley Publishing Company, 1984.

[3] Stanley Morison, First principles of typography, Cambridge University Press, 1936.

[4] Leslie Lamport, LATEX, a document preparation system, Addison-Wesley Publishing Company, 1986.

◇ Victor Eijkhout
Center for Supercomputing
Research and Development
University of Illinois
305 Talbot Laboratory
104 South Wright Street
Urbana, Illinois 61801-2932, USA
eijkhout@csrd.uiuc.edu

So, in order to suppress expansion in a ⟨*token string*⟩ which is to be written out to an external file, one need only stuff the ⟨*token string*⟩ into a macro, regurgitate the macro's `\meaning` into another macro, and `\write` the second macro out. The data being written out is, in a certain sense, inert because control sequences have been divided into the characters forming their names and there is nothing to be expanded. If this information is reread from an external file again, however, (and therefore passes through TEX's mouth again) it can be reassembled into the original ⟨*token string*⟩.

To this end, we make the following definition:

```
\def\GetMacroMeaning#1:->#2:->#3\endget{%
  \def#3{#2}}
```

The third argument to `\GetMacroMeaning` is the control sequence into which the "meaning" will be placed. The first two arguments will be produced by 'expanding' `\meaning`. Thus, converting text for, say, a section head which a user keys as `\section{...}`, can be accomplished by inserting the following sort of code within the definition of `\section`:

```
\def\section #1{%
  ...
  \def\sectionhead{#1}%
  \expandafter\GetMacroMeaning
    \meaning\sectionhead
    :->\xxsectionhead\endget
  \write\outfile
    \expandafter{\xxsectionhead}%
  ...
  }
```

The first `\expandafter` causes `\meaning` to gobble `\sectionhead` and expand into the first 2 arguments of `\GetMacroMeaning`. The second `\expandafter` is used in the `\write` statement because the code presented always stores the head to be written out in `\xxsectionhead`. When 2 section heads occur on the same page, the second will overwrite the first definition of `\xxsectionhead`, so we must make sure that the contents of `\xxsectionhead` gets placed in the node on the page and not just the token `\xxsectionhead` itself. In situations where one may write `\immediately`, the line under discussion could become

```
\immediate\write\outfile{\xxsectionhead}%
```

Further Problems

The method above concerns itself only with material we wish to block from expansion as we `\write` it out. As noted previously, other data (such as page and section numbers) must be expanded to get tables of contents and indexes right. David Salomon has discussed some of these issues in *TUGboat* 10, no. 3. For this article, we only point out that one can concatenate different kinds of data into one control sequence and then `\write` that out. For example, if `\sectionnumber` is a TEX count register containing the current section number, one might augment and alter the above code to

```
\def\sectag{\sec}%
\expandafter\GetMacroMeaning
   \meaning\sectag
   :->\xxsectag\endget

\def\section #1{%
  ...
  \def\sectionhead{#1}%
  \expandafter\GetMacroMeaning
     \meaning\sectionhead
     :->\xxsectionhead\endget
  \edef\writedata{%
      \xxsectag
      {\number\sectionnumber}%
      {\xxsectionhead}%
      {\noexpand\number\pageno}%
    }%
  \write\outfile\expandafter{\writedata}%
  ...
  }
```

Thus, for section number 3 with title "On $\sin^2 x + \cos^2 x = 1$" and appearing on page 22, the above code will cause

```
\sec {3}{On $\sin ^2x+\cos ^2x=1$}{22}
```

to be written on `\outfile`. The `\edef` for `\writedata` causes expansion in its replacement text where possible, so the definition of `\writedata` in the case above will be

```
\sec {3}{$\sin ^2x+\cos ^2x=1$}{\number\pageno}
```

Of course, `\sec` here consists of 4 tokens of category 12 (since the whole line passed through `\meaning`), not just one control sequence, and a similar remark holds for the text of the section head. `\number\pageno`, however, has not been sanitized and will be expanded as this token string is written to `\outfile`.

A considerable compaction of all this code can be had by doing the expansions at once (as suggested by Victor Eijkhout). To this end, make the definitions

```
\def\gobble#1:->{}
\def\sanitize{%
   \expandafter\gobble\meaning}
```

```
\def\sectag{\sec}
```

and then rework \section to

```
\def\section #1{%
  ...
  \def\sectionhead{#1}%
  \edef\act{%
    \write\outfile{%
      \sanitize\sectag
      {\number\sectionnumber}%
      {\sanitize\sectionhead}%
      {\noexpand\number\pageno}%
      }}%
  \act
  ...
  }
```

Another problem occurs when one wishes to \write out long strings of text. Peter Breitenlohner showed in *TUGboat* 11, no. 1 that one could break long strings of text exactly as they had been broken in the source file by activating ⟨*carriage return*⟩s and specifying the \newlinechar to be the ⟨*carriage return*⟩. Unfortunately, this method is not transferrable within the current technique exactly because of the sanitizing properties of \meaning. \meaning will disgorge ^^M sequences for active ⟨*carriage return*⟩s which cannot in turn be read as ⟨*carriage return*⟩s because the ^ will be of category 'other' instead of 'superscript'.

One way to get over this is to use active ^^Ms as delimiters of line records and \write the intervening material line by line to the output file. Here we use a method of Alois Kaelschacht (*TUGboat* 8, no. 2, p. 184; also pointed out by Sonja Maus recently) which allows \loops to contain \else clauses.

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

To handle a long piece of text line by line, we define \ParseLine to divide the material after it into 2 pieces separated by the first ⟨*carriage return*⟩ in that material.

```
{\catcode`\^^M=\active
\gdef\ParseLine#1^^M#2\endParse{%
  \def\FirstLine{#1}%
  \def\Remainder{#2}%
  }}
```

\Writeit first turns on the ⟨*carriage return*⟩, then reads the text to be written to a file. Then it runs through a \loop until the \Remainder text is empty, writing each line with the \meaning technique.

```
{\endlinechar=-1
\catcode`\^^M=\active
\catcode`\@=11
\gdef\Writeit{
  \bgroup\catcode`\^^M=\active
  \@Writeit}

\gdef\@Writeit#1\endWriteit{
  \let\FirstLine\empty
  \def\Remainder{#1^^M}
  \loop
    \expandafter
      \ParseLine\Remainder\endParse
    \write\outfile\expandafter{
                \sanitize\FirstLine}
  \ifx\Remainder\empty\else\repeat
  \egroup}}
```

Of course, this is exactly the kind of nonsense that Breitenlohner avoided with the \newlinechar technique, and the 'solution' here still has problems. For one thing, pairs of braces which occur across input lines will cause \ParseLine to miss the intervening ^^Ms (since arguments to macros must contain balanced sets of braces). For another, if one also wishes to use the argument to \Writeit for something else (such as typesetting here and now), the ^^Ms are now embedded and have not been changed into ⟨*space*⟩ or \par as appropriate. It is possible to define the active ^^M to check ahead for another ^^M immediately following, changing the pair into \par and otherwise inserting ⟨*space*⟩, but at this point we realize we are trying to simulate TEX's mouth behavior with a stomach process and will never be wholly successful. When an input line ends with a control word, TEX's mouth will gobble the end-of-line character, whereas the procedure above will insert an end-of-line ⟨*space*⟩ willy-nilly.

All of which is to say that the \meaning technique outlined here should be confined to cases where one is fairly certain that the records to be written are rather short (say, the cases of tables of contents and indexes). In cases where longer records are anticipated, Breitenlohner's technique, accompanied by something analogous to \protect, is needed; or one may simply use a verbatim approach if the data is not to be used 'immediately'.

◇ Ron Whitney
TEX Users Group
rfw@Math.AMS.com