- There is no integrated text editor. OzTEX is distributed with ΣEdit, a public domain DA editor written by Leonard Rosenthol.
- OzTEX requires a PostScript printer.
- \special handling is fairly unsophisticated. OzTEX allows the inclusion of a PostScript file along with optional code prefixed to the file. There is currently no support for previewing PICT or EPSF files.
- Previewing DVI pages is not as fast as I'd like, particularly on a Mac Plus.

Future development of OzTEX is likely but will occur at a fairly sedate pace unless I can find people prepared to help with the programming or provide financial support. Send your bug reports, comments and offers of help to the address shown at the end of this article.

Where to get OzTEX

The following people have volunteered to help distribute OzTEX. Please get in touch with the person nearest you. By the time you read this article it is likely that OzTEX will also be available electronically from various Mac archive sites. People without access to email should try their local Mac user group.

In Australia and New Zealand:

addie@rhea.trl.oz	Ron Addie, Melbourne
keady@madvax.uwa.oz	Grant Keady, Perth
rks105@phys6.anu.oz	Russell Standish, Canberra
ccc032u@aucc4341.aukuni.ac.nz	R. Fulton, Auck.

In the USA:

c3ar@zaphod.uchicago.edu	Walter Carlip, Chicago
tnieland@aamrl.af.mil	Ted Nieland, Dayton
spencer@cis.ohio-state.edu	S. Spencer, Columbus

In the UK and Europe:

abbottp@aston.ac.uk	Peter Abbott, UK
texline@vaxa.cc.ic.ac.uk	Malcolm Clark, UK
nikunen@cc.helsinki.fi	Martti Nikunen, Helsinki

I'd like to hear from people interested in distributing OzTEX in other countries. Here's how to get in touch:

Andrew Trevorrow
Kathleen Lumley College
North Adelaide, SA, 5006, Australia
Telephone: (08) 267 1060
Email: atrevorrow@g.ua.oz (ACSnet)

Tutorials

\string and \csname

Stephan v. Bechtolsheim

This article discusses \string and \csname to convert back and forth between strings and tokens. To control loading macro source files in a convenient way, I will show an application of \csname. I will also discuss cross referencing which relies on \csname.

Converting Tokens to Strings, \string

"\string <token>" causes TEX to read the token <token> following \string without expansion. Subsequently <token> is replaced by a string representing it. Let me start with some examples.

1. {\tt\string\hskip} prints \hskip.
2. {\tt\string$} prints $.
3. {\tt\string\$} prints \$.
4. {\tt\string{} prints {.
5. {\tt\string}} prints }.

Also note:

1. The escape character printed in the previous examples is the backslash. Any other character could be printed by assigning a different character code to \escapechar. The default is obviously \escapechar = `\\, which assigns the character code of the backslash. If you change \escapechar to a negative value, then no escape character is printed:
   \escapechar = -1 \string\xx prints xx.
2. There is an important difference between 'xx' entered as an ordinary string and 'xx' generated using \string as just shown. All characters generated by \string have the category code 12 ("other"), whereas 'x' ordinarily has category code 11 ("letter").
3. Observe the use of the typewriter font (\tt). If you use the roman font and simply write \string\hskip the output reads "hskip and *not* \hskip, as expected. The reason for this is that the roman font contains an opening double quote in the position where the typewriter font contains a backslash.
4. \string converts only the token following it into a string. For instance, to print two consecutive $$ you have to repeat \string and enter {\tt\string$\string$}. If you enter only {\tt\string$$} the first dollar sign is

printed due to \string, the second one causes TEX to enter math mode.

5. An important application of \string is to write control sequences to a file using \write. Any control sequence which should be written to a file (instead of being expanded) must be prefixed by \string. \noexpand can also be used.

Converting Strings into Tokens, \csname ... \endcsname

General Discussion. The \csname instruction is, in a certain sense, the inverse operation of \string. It converts a sequence of *characters* into one token. Observe that I said "characters" and not "letters." Using \csname allows you to build names for tokens that contain nonletter characters such as digits. The ordinary way to write control sequences restricts the user to control words (the escape character followed by any number of letters, but letters only) and control symbols (the escape character followed by one and only one nonletter character).

The \csname control sequence is applied as follows. After \csname, list the characters naming the token. You also may use macros, but only those which expand to characters. The sequence of characters forming the name of the token is terminated by \endcsname.

Here is an example. To name the token "\?-a*17.g" write

```
\csname ?-a*17.g\endcsname
```

In the rest of this article, please allow a certain looseness with respect to notation. Ordinarily, if you see a piece of TEX source like \?-a*17.g, you would interpret this as \? followed by 7 single character tokens. In this article, it stands for one single token.

Depending upon the context of the above example, TEX may try to expand the token named and will have to be able to find a corresponding macro definition (or any other type of definition). Here is how one can define such a token.

```
\expandafter\def
    \csname ?-a*17.g\endcsname{%
        Replacement text of macro.
}
```

The \expandafter suppresses the \def temporarily to allow TEX to compute the name of the token \?-a*17.g. Then \def is re-inserted in front of this token. The macro definition now proceeds as any other macro definition. If \expandafter were omitted, TEX would define a macro with the name \csname.

As mentioned before it is legal to call a macro inside a \csname ... \endcsname sequence as long as the macro expands to characters only. Counter-registers can also be used:

```
\def\xx{ABC}
\count0 = 4
\csname ZZ-\the\count0-\xx\endcsname
```

This example is equivalent to forming the same token using \csname ZZ-4-ABC\endcsname.

I will later discuss two applications of \csname. One will define a macro \InputD to load macro source files, and the other uses \csname for cross-referencing macros.

\csname and \relax. Assuming that *no* preceding definition for \xx is given, when TEX executes the following code:

```
\csname xx\endcsname
\xx
```

You may be surprised to see that the first line does *not* generate an "undefined control sequence" error, whereas the second line does. The reason is that *undefined* tokens generated by \csname are made equivalent to \relax by TEX. The above code fragment is therefore equivalent to:

```
\relax
\xx
```

On the other hand, in the following examples:

```
\def\xx{This is fun}
\csname xx\endcsname
\xx

\expandafter\def\csname xx\endcsname{%
    This is fun%
}
\csname xx\endcsname
\xx
```

"This is fun" is printed four times.

In other words, if a token named using \csname is undefined, it is equivalent to \relax. This fact will be used in definition of \InputD (next section).

\InputD: Loading Macros Conveniently

The Problem. I personally like to divide my macro sources into many small macro source files. I load only those macro source files that I really need. Here is a problem frequently encountered in this context:

1. Assume that at the top of a main source file, main.tex, you load macro file A.tex. This

macro file in turn uses macros from another macro source file `B.tex`.

2. Assume in addition that you load macro source file `C.tex` in `main.tex`, and that `C.tex` also uses macros from `B.tex`.

If you load both `A.tex` and `C.tex` in your main source file, and both `A.tex` and `C.tex` load `B.tex` using `\input`, then, of course, `B.tex` will be loaded twice, which is undesirable.

Using `\InputD`. I will now explain how to define a macro `\InputD` having one argument, the name of a file, which loads the file only if the file was not loaded before. In other words, in `A.tex` and `C.tex` you request `B.tex` to be loaded via `\InputD{B.tex}`. Only the very first time is `\InputD{B.tex}` equivalent to `\input B.tex`; subsequent times, `\InputD{B.tex}` does nothing (actually, in the macro definition below, a message is generated saying that `B.tex` was not loaded again). The macro `\InputD` does all the bookkeeping.

Note that you must always use this macro to load macro source files in order to get the effects described here. After using the ordinary `\input B.tex` (bypassing the bookkeeping of `\InputD`), TEX has no record of the fact that `B.tex` was already loaded, and a later occurrence of `\InputD{B.tex}` will cause TEX to load `B.tex` again if this is the first call of `\InputD` with argument `B.tex`.

The Workings of `\InputD`. When this macro is called, its argument, a file name, will be used to form a token as follows. A prefix, `InputD-`, and the file name will be concatenated. For instance, `\InputD{B.tex}` causes TEX to form the token `\InputD-B.tex`. A test is then performed to determine whether this token is already defined. If it is (which will mean the file `B.tex` is already loaded), nothing is done. On the other hand, if this token is *undefined* (this is true only the very first time `\InputD` is called with argument `B.tex`), the macro will *define* token `\InputD-B.tex` and load file `B.tex`. Any subsequent call `\InputD{B.tex}` will find the token `InputD-B.tex` defined. The actual definition of this token is irrelevant. The macro `\InputD` simply defines the token to expand to nothing (i.e., the replacement text is empty).

I think that using this macro offers a very flexible and powerful approach to maintaining lots of macro source files. There is no longer a need to do any bookkeeping of which source files have been loaded: if you need a macro source file, load it using `\InputD`. Source files will be loaded only if they have not been loaded before.

The Definition of `\InputD`. The definition of the `\InputD` macro is amazingly simple. Note that `\ifx` absorbs without expansion the two tokens following it, and then compares the two tokens. `\expandafter` is used first to compute the token `InputD-B.tex` (assuming the argument is `B.tex`), and then the `\ifx` compares this token with `\relax`. As noted before, if the token is undefined, it is equivalent to `\relax`.

```
\def\InputD #1{%
    \expandafter\ifx
        \csname InputD-#1\endcsname
        \relax
        % Equivalent to \relax: not
        % defined before! Define now.
        \expandafter\def
            \csname InputD-#1%
                \endcsname{}%
        % Read in macro source file.
        \input #1
    \else
        % Loaded already.
        % Print message.
        \message{\string\InputD:
            file "#1" was loaded
            before.}%
    \fi
}
```

Cross-Referencing Macros

The following discussion is more complicated than the previous application for `\csname`. This is a brief sketch only (see my book "TEX in Practice" for further information).

The User Interface. Let me first explain how you can use the cross-reference macros which I will present later. The following should be familiar to every user of LATEX, the main difference being that I use macros which begin with capital letters. To identify document entities such as chapters, sections, figures, etc., *labels* are used. Such labels consist of arbitrary characters like `f-structure` for a figure describing the structure of some piece of equipment.

The user has the following three macros at her/his disposal (all three have one parameter, a label):

1. `\Label` is used to define a label. For instance, `\Label{f-structure}` labels the figure mentioned above, associating a symbolic reference ("`f-structure`") with an appropriate numeric reference (say, "3.5").

2. \Ref expands to the figure number of the figure whose label is given as an argument. If you want to print some text like "see Fig. 3.5," then you would *not* enter "see Fig.~3.5," because you would have to change this text if the figure number changes (for instance, to 3.6, because you have inserted another figure before this figure). Instead you enter "see Fig.~\Ref{f-structure}," and let TEX do the work.
3. \PageRef expands to the page number of the figure labeled by the name you provide. Again this page number will automatically change, if the figure migrates to a different page due to modifications of the text. Your input might read "p.~\PageRef{f-structure}" and print "p. 67" if that is the page where the figure with the specified label is placed.

Retrieving the Cross-Reference Information. The cross-reference information is read in at the very beginning of a TEX job (before any "real" text processing is started) from a *label file*. In the case of LATEX, this label information is stored in .aux files. We will soon discuss how this information was written to the label file in the first place.

Such label files consist of calls to a macro \NewLabel. This macro has three arguments: *label name*, *number* of the entity and *page number* of the entity. So in the above case one call contained in this label file reads as follows:

```
\NewLabel{f-structure}{3.5}{67}
```

\NewLabel, when called, will in turn define two macros, one called \REF-f-structure, which expands to the number of this figure, and one called \PAGEREF-f-structure, which expands to the page number. Here is the definition of \NewLabel.

```
\def\NewLabel #1#2#3{%
  \expandafter\def
    \csname REF-#1\endcsname{#2}%
  \expandafter\def
    \csname PAGEREF-#1\endcsname{#3}%
}
```

\Ref and \PageRef simply retrieve what \NewLabel has stored:

```
\def\Ref #1{\csname REF-#1\endcsname}
\def\PageRef #1{%
    \csname PAGEREF-#1\endcsname}
```

Note that a definition of \Ref or \PageRef would normally be augmented by an \ifx test along the lines of the definition of \InputD to print a warning message in the case of an undefined label.

Generating the Label File. From what we discussed so far you know that the label file is read in at the very beginning of a TEX run. Therefore *all* cross-references printed in the text are based on the information of the *previous* run. They do *not* take into account any changes occurring to those labels due to changes in the text during the current run. The same file name is used in the current run both for reading the old label file and for writing new label information in the new version of the label file.

In the definition of \Label below it is assumed that \TheFigureNumber produces the current figure number. This macro has one parameter which is the name of the label. \LabelStream is assumed to be the stream for writing the label file.

```
\def\WriteLab{\write\LabelStream}
\def\Label #1{%
    \edef\LabelTemp{%
        \noexpand\string
        \noexpand\NewLabel
            {#1}{\TheFigureNumber}}
    \expandafter\expandafter\expandafter
        \WriteLab\expandafter{%
            \LabelTemp{\the\pageno}}%
}
```

Features Not Discussed. I only tried to sketch here how cross-referencing macros can be implemented. The above macros are far from complete. The following details were ignored.

1. Opening and closing the label file. Reading in the label file in the very beginning.
2. Printing a warning message if two figures are accidentally labeled by the same label.
3. Using labels to label other entities like chapters, sections, tables, etc.
4. Printing a warning message if label definitions as generated during the current run differ from label definitions of the previous run, resulting in possibly wrong cross-references and requiring processing a document for a second time.

Concluding Remarks

This article was derived from my book "TEX in Practice" (the original title was "Another Look at TEX"). The book will be published by Springer in October of this year.

◇ Stephan v. Bechtolsheim
Integrated Computer Software, Inc.
2119 Old Oak Drive
West Lafayette, IN 47906
svb@cs.purdue.edu