IdxTEX and GloTEX Indexes and Glossaries

Richard L. Aurbach
Monsanto Company

With reference to Jim Ludden's request for a program to format LATEX indices, we have developed two programs here at Monsanto which you may find interesting.

IdxTEX

The IdxTEX program is an automatic index generator for LATEX. It features three-level indexing, visual highlighting of index entries and page number references, and generates a file which may be `\input` into your document to produce a fully-automated and fully-formatted index.

GloTEX

The GloTEX program is to glossaries what BIBTEX is to bibliographies — it is a program which uses databases of definitions to produce nicely-formatted glossaries, using the `\makeglossary` and `\glossary` features of standard LATEX.

Both of these programs are being made available to TUG for distribution to interested parties. The programs are VAX/VMS specific, but are written in C and are (I hope) exhaustively documented. They are not portable (because they call VAX/VMS services), but should be relatively easy to port to other environments. The distribution kit contains executable images, full sources, users' guides (and the special document styled needed to generate them), and on-line help modules.

I request that anyone who enhances these programs, finds and fixes bugs, or makes any other changes in these programs (other than porting them to other environments) let me know so we can also benefit. I would also be interested in ports to the VM/CMS environment.

Queries

Editor's note: When answering a query, please send a copy of your answer to the TUGboat editor as well as to the author of the query. All answers will be published in the next issue following their receipt.

In addition to the item below on change bars (query by Sylvester Fernandez, Vol. 7, No. 2, page 110), two responses to the query by Jim Ludden (*ibid.*, page 111) regarding post-LATEX index formatting appear in the LATEX column, beginning on page 186.

Form Letters

Is there a package available that allows the generation of identical letters *except* for the addressee field, which is read from a separate file containing lines of address separated by a delimiter?

John Lee
`jslee@nrtc.arpa`

Change Bars

Jim Fox
University of Washington

The question of change bars was asked in the last TUGboat. Here is how we do them at the Academic Computing Center.

start change bar:

```
\special{changebar, \the\barwidth}
```

end change bar:

```
\special{changebar}
```

where `\barwidth` is a ⟨dimen⟩ register that describes the width of the bar. A vertical rule will be drawn from the location of the first special to the vertical coordinate of the second special. Note that the second special only defines the vertical extent of the rule — its horizontal coordinate is ignored.

Macros that are compatible with `plain.tex` output routines do the positioning automatically —

the user need only type \beginbar and \endbar. Marks are used to stop and start multiple page change bars.

The change bar question brings up some interesting points.

A problem arises when you also want to use marks for something other than change bars: chapter numbers, for example. In that case you can't just have \beginbar include, say, \mark{\startabar} because you would lose the chapter number information that was also being kept in mark text.

I haven't implemented a general solution to this problem, but I think it could go as follows. Define a \newmark macro that would be invoked for each distinct mark function. In this case \newmark\barmark and \newmark\chaptmark. Then provide a \setmarks macro that defs each of the allocated marks; e.g., in this case \setmarks would be (automatically) defined as follows:

```
\def\setmarks{\mark{%
   \def\noexpand\thebarmark{\barmark}%
   \def\noexpand
            \thechaptmark{\chaptmark}}}
```

then in the text, the usage is

```
\def\barmark{\startabar}\setmarks
```

and

```
\def\chaptmark{...}\setmarks.
```

In the output routine the appropriate marks are first defined and then used:

```
\botmark ...
```

followed by

```
\thebarmark ... and \thechaptmark ...
```

The idea is that the actual mark contains only \defs, which are defined when \botmark (or \topmark, etc.) is referenced.

The second point concerns \specials in general. It does not seem to be universally understood that the random paging mechanism in the dvi file format implicitly proscribes global specials [cf. TUGboat 6, #2 pp. 66–69]. Any global formatting function that uses specials (changing the paper orientation, for example) must repeat the appropriate special command on every output page.

In addition, dvi file printers should be careful not to remember \special parameters between pages.

Letters

Bugs in METAFONTware

To the Editor:

I have discovered a couple of bugs in the METAFONT utility programs having to do with packed files and would like to share this discovery.

The first bug is severe, and makes it virtually impossible to use packed files. It occurs in the Kellerman and Smith implementation (VAX) of PKtoPX (version 2.2), the program which converts packed files to the PXL format most commonly used by device drivers. The bug is in the change file rather than the WEB file, so none of the other implementations are affected. I don't know whether this bug has been previously discovered or not; the number of sites using PK files is still limited. Also, if a driver reads PK files directly, it does not use PKtoPX and the bug does not apply.

The nature of the bug is that, in the Font Directory at the end of the PXL file, the pointers to the glyphs are incorrectly expressed, making it impossible for the driver to find the rasters for the glyph in the main part of the file. According to section 9 of PKtoPX, "The third word of a glyph's directory information contains the number of the word in this PXL file where the raster description for this particular glyph begins, measured from the first word which is numbered zero." Word, in this context, refers to a 32-bit number ("longword" in VAX terminology). The problem is that the changes for the VAX implementation accidentally change this offset from an offset of longwords to an offset of bytes, making the offsets four times greater than they should be. The procedure in question, *pixel_integer*, writes a 32-bit integer to the PXL file and increments a variable called *pxl_loc*, which contains the current offset into the PXL file in longwords. Kellerman and Smith changed this procedure to write the integer as four separate bytes, which is all well and good, but logically the PXL file is still a stream of longwords, so the increment of *pxl_loc* should have been left alone.

To fix the bug, find the following line in PKTOPX.CH:

```
pxl_loc:=pxl_loc+4;
```

and change it to:

```
incr(pxl_loc) ;
```

This will force the variable *pxl_loc* to once again be a longword-count instead of a byte-count.