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.

The other bug, which occurs in GFtoPK (version 1.2), is more subtle, and possibly inconsequential, depending on the driver which is using the packed file. Unfortunately, the error is in the main WEB file, causing it to crop up in *all* implementations.

The problem occurs in module 64, "⟨Scan for bounding box⟩". This code is supposed to take the pixel image of a glyph as generated by METAFONT and strip off rows and columns consisting of all-white pixels from the sides, top, and bottom, resulting in a tighter character. It seems to work in all cases except when there are one or more all-white columns on the left. When this happens, the algorithm fails and the *max_m* variable (and therefore *width*) is set to a value one greater than it should be. PKtoPX cheerfully passes this incorrect value to the font directory in the PXL file. Whether this is a problem or not depends on the individual device driver. The actual raster image is correct — it's just the width that is wrong. If the driver uses the width in the TFM file rather than the one in the PXL file, then the error has absolutely no effect. If the width in the PXL file is used, the character appears one pixel wider than it actually is. In any case, not many characters are affected. I am not familiar enough with METAFONT to describe the conditions which cause all-zero columns to be added on the left. It appears on various characters in various fonts at various magnifications. For an example, do a GFtype of cmr10.300GF and look at the character with a decimal code of 20 (ˇ). After suppression opf the left column, which is all zeros, the actual width of the character should be 10, but a PKtype after GFtoPK is run will show a width of 11. This is the only character in this particular font file which has this problem; other font files have more than one erroneous character, while still others have none at all.

To fix this bug, add the following section to GFTOPK.CH:

```
@x
  min_m := min_m + extra ;
  max_m := min_m + max_m - 1 ;
  height := max_n - min_n + 1 ;
  width := max_m - min_m + 1 ;
@y
  min_m := min_m + extra ;
  max_m := min_m + max_m - 1 - extra ;
  height := max_n - min_n + 1 ;
  width := max_m - min_m + 1 ;
@z
```

Where this section goes in the change file depends on what changes are already in the file, of course. For the Kellerman and Smith VAX version, it goes immediately ahead of:

```
@x
@d preamble_comment == 'GFtoPK 1.2 output'
@d comm_length = 17
@y
@d preamble_comment ==
            'VAX/VMS GFtoPK 1.2.0 output'
@d comm_length = 27
@z
```

E. W. Sewell
Software Engineering Specialist
3822 Hillsdale Lane
Garland, TX 75042-5334
(214) 272-0515

Editor's note: These problems have been communicated to David Kellerman at K&S and to Tom Rokicki at Stanford. David agrees that use of PK files is still very limited, and therefore these programs haven't been given a great amount of exercise. He was already aware of both bugs and said that all corrections would be on the K&S distribution tape no later than October 1, 1986.

Tom Rokicki responded with additional updates to the PK file software; see his note on page 140.