Software

The New Versions of TEX and METAFONT

Donald E. Knuth

For more than five years I held firm to my conviction that a stable system was far better than a system that continues to evolve. But during the TUG meeting at Stanford in August, 1989, I was persuaded to make one last set of changes, in order to bring TEX and METAFONT to a state of completion consistent with their overall philosophy and goals.

The main reason for the changes was the fact that I had guessed wrong about 7-bit character sets versus 8-bit character sets. I believed that standard text input would continue indefinitely to be confined to at most 128 characters, since I did not think a keyboard with 256 different outputs would be especially efficient. Needless to say, I was proved wrong, especially by developments in Europe and Asia. As soon as I realized that a text formatting program with 7-bit input would rapidly begin to seem as archaic as the 6-bit systems we once had, I knew that a fundamental revision was necessary.

But the 7-bit assumption pervaded everything, so I needed to take the programs apart and redo them thoroughly in 8-bit style. This put TEX onto the operating table and under the knife for the first time since 1984, and I had a final opportunity to include a few new features that had occurred to me or been suggested by users since then.

The new extensions are entirely upward compatible with previous versions of TEX and METAFONT (with a few small exceptions mentioned below). This means that error-free inputs to the old TEX and METAFONT will still be error-free inputs to the new systems, and they will still produce the same outputs.

However, anybody who dares to use the new extensions will be unable to get the desired results from old versions of TEX and METAFONT. I am therefore asking the TEX community to update all copies of the old versions as soon as possible. Let us root out and destroy the obsolete 7-bit systems, even though we were able to do many fine things with them.

In this note I'll discuss the changes, one by one; then I'll describe the exceptions to upward compatibility.

1. The character set

Up to 256 distinct characters are now allowed in input files. The codes that were formerly limited to the range `0..127` are now in the range `0..255`. All characters are alike; you are free to use any character for any purpose in TEX, assigning appropriate values to its `\catcode`, `\mathcode`, `\lccode`, `\uccode`, `\sfcode`, and `\delcode`. Plain TEX initializes these code values for characters above 127 just as it initializes the codes for ordinary punctuation characters like '`!`'.

There's a new convention for inputting an arbitrary 8-bit character to TEX when you can't necessarily type it: The four consecutive characters `^^`$\alpha\beta$, where α and β are any of the "lowercase hexadecimal digits" `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `a`, `b`, `c`, `d`, `e`, or `f`, are treated by TEX on input as if they were a single character with specified code digits. For example, `^^80` gives character code 128; the entire character set is available from `^^00` to `^^ff`. The old convention discussed in Appendix C, under which character 0 was `^^@`, character 1 (control–A) was `^^A`, ..., and character 127 was `^^?`, still works for the first 128 character codes, except that the character following `^^` should not be a lowercase hexadecimal digit when the immediately following character is another such digit.

The existence of 8-bit characters has less effect in METAFONT than in TEX, because METAFONT's character classes are built in to each installation. The normal set of 95 printing characters described on page 51 of *The METAFONTbook* can be supplemented by extended characters as discussed on page 282, but this is rarely done because it leads to problems of portability. METAFONT's **char** operator is now redefined to operate modulo 256 instead of modulo 128.

2. Hyphenation tables

Up to 256 distinct sets of rules for hyphenation are now allowed in TEX. There's a new integer parameter called `\language`, whose current value specifies the hyphenation convention in force. If `\language` is negative or greater than 255, TEX acts as if `\language` = 0.

When you list hyphenation exceptions with TEX's `\hyphenation` primitive, those exceptions apply to the current language only. Similarly, the `\patterns` primitive tells TEX to remember new hyphenation patterns for the current language; this operation is allowed only in the special "initialization" program called `INITEX`. Hyphenation exceptions can be added at any time, but new patterns cannot be added after a paragraph has been typeset.

When TeX reads the text of a paragraph, it automatically inserts "whatsit nodes" into the horizontal list for that paragraph whenever a character comes from a different `\language` than its predecessor. In that way TeX can tell what hyphenation rules to use on each word of the paragraph even if you switch frequently back and forth among many different languages.

The special whatsit nodes are inserted automatically in unrestricted horizontal mode (i.e., when you are creating a paragraph, but not when you are specifying the contents of an hbox). You can insert a special whatsit yourself in restricted horizontal mode by saying `\language`⟨number⟩. This is needed only if you are doing something tricky, like unboxing some contribution to a paragraph.

3. Hyphenated fragment control

TeX has new parameters `\lefthyphenmin` and `\righthyphenmin`, which specify the smallest word fragments that will appear at the beginning or end of a word that has been hyphenated. Previously the values `\lefthyphenmin=2` and `\righthyphenmin=3` were hard-wired into TeX and impossible to change. Now plain TeX format supplies the old values, which are still recommended for most American publications; but you can get more hyphens by decreasing these parameters, and you can get fewer hyphens by increasing them. If the sum of `\lefthyphenmin` and `\righthyphenmin` is 63 or more, all hyphenation is suppressed. (You can also suppress hyphenation by using a font with `\hyphenchar=-1`, or by switching to a `\language` that has no hyphenation patterns or exceptions.)

4. Smarter ligatures

Now here's the most radical change. Previous versions of TeX had only one kind of ligature, in which two characters like 'f' and 'i' were changed into a single character like 'fi' when they appeared consecutively. The new TeX understands much more complex constructions by which, for example, we could change an 'i' following 'f' to a dotless 'ı' while the 'f' remains unchanged: 'fı'.

As before, you get ligatures only if they have been provided in the font you are using. So let's look at the new features of METAFONT by which enhanced ligatures can be created. A METAFONT programmer can specify a "ligature/kerning program" for any character of the font being created. If, for example, the 'fi' combination appears in font position 12, the replacement of 'f' and 'ı' by 'fi' is specified by including the statement

```
"i" =: 12
```

in the ligature/kerning program for "f"; this is METAFONT's present convention.

The new ligatures allow you to retain one or both of the original characters while inserting a new one. Instead of `=:` you can also write `|=:` if you wish to retain the left character, or `=:|` if you wish to retain the right character, or `|=:|` if you want to keep them both. For example, if the dotless ı appears in font position 16, you can get the behavior mentioned above by having

```
"i" |=:  16
```

in f's program.

There also are four additional operators

```
|=:>,    =:|>,    |=:|>,    |=:|>>,
```

where each > tells TeX to shift its focus one position to the right. For example, if f and i had been replaced by f and dotless ı as above, TeX would begin again to execute f's ligature/kern program, possibly inserting a kern before the dotless ı, or possibly changing the f to an entirely different character, etc. But if the instruction had been

```
"i" |=:> 16
```

instead, TeX would turn immediately to the ligature/kern program for characters following character 16 (the dotless ı); no further change would be made between f and ı even if the font had something specified there.

5. Boundary ligatures

Every consecutive string of 'characters' read by TeX in horizontal mode (after macro expansion) can be called a 'word'. (Technically we consider a 'character' in this definition to be either a character whose `\catcode` is a letter or otherchar, or a control sequence that has been `\let` equal to such a character, or a control sequence that has been defined by `\chardef`, or the construction `\char`⟨number⟩.) The new TeX now imagines that there is an invisible "left boundary character" just before every such word, and an invisible "right boundary character" just after it. These boundary characters take effect if the font designer has specified ligatures and/or kerning between them and the adjacent letters. Thus, the first or last character of a word can now be made to change its shape automatically.

A ligature/kern program for the left boundary character is specified within METAFONT by using the special label `||:` in a **ligtable** command. A ligature or kern with the right boundary character is specified by assigning a value to the new internal METAFONT parameter *boundarychar*, and by specifying a ligature or kern with respect to this character. The *boundarychar* may or may not exist as a real character in the font.

For example, suppose we want to change the first letter of a word from 'F' to 'ff' if we are doing some olde English. The METAFONT font designer could then say

```
ligtable ||:  "F"  |:= 11
```

if character 11 is the 'ff'. The same ligtable instruction should appear in the programs for characters like (and ' and " and - that can precede strings of letters; then '`Bassington-French`' will yield 'Bassington-ffrench'.

If the 's' of our font is the pre-19th century s that looks like a mutilated 'f', and if we have a modern 's' in position 128, we can convert the final s's as Ben Franklin did by introducing ligature instructions such as

```
boundarychar := 255;
ligtable "s":  255 =:| 128,
               "." =:| 128,
               "," =:| 128,
               ")" =:| 128,
               "'" =:| 128,
```

and so on. (A true oldstyle font would also have ligatures for ss and si and sl and ssi and ssl and st; it would be fun to create a Computer Modern Oldstyle.)

The implicit left boundary character is omitted by TeX if you say `\noboundary` just before the word; the implicit right boundary is omitted if you say `\noboundary` just after it.

6. More compact ligatures. Two or more ligtables can now share common code. To do this in METAFONT, you say '**skipto** ⟨n⟩' at the end of one **ligtable** command, then you say '⟨n⟩::' within another. Such local labels can be reused; e.g., you can say **skipto** 1 again after `1::` has appeared, and this skips to the *next* appearance of `1::`. There are 256 local labels, numbered 0 to 255. Restriction: At most 128 ligature or kern commands can intervene between a **skipto** and its matching label.

The `TFM` file format has been upwardly extended to allow more than 32,500 ligature/kern commands per font. (Previously there was an effective limit of 256.)

7. Better looking sloppiness

There is now a better way to avoid overfull boxes, for people who don't want to look at their documents to fix unfeasible line breaks manually. Previously people tried to do this by setting `\tolerance=10000`, but the result was terrible because TeX would tend to consolidate all the badness in one truly horrible line. (TeX considers all badness $\geq$ 10000 to be infinitely bad, and all these infinities are equal.)

The new feature is a dimension parameter called `\emergencystretch`. If `\emergencystretch` is positive and if TeX has been unable to typeset a paragraph without exceeding the given tolerances, another pass over the paragraph is made in which TeX pretends that additional stretchability equal to `\emergencystretch` is present in every line. The effect of this is to scale down all the badnesses into a range where previously infinite cases become finite; TeX will find an optimum solution to the scaled-down problem, and this will be about as good as possible in a practical sense. (The extra stretching is not really present; therefore underfull boxes will be reported in warning messages unless `\hbadness` is increased.)

8. Looking at badness

TeX has a new internal integer parameter called `\badness` that records the badness of the box it has most recently constructed. If that box was overfull, `\badness` will be 1000000; otherwise `\badness` will be between 0 and 10000.

9. Looking at the line number

TeX also has a new internal integer parameter called `\inputlineno`, which contains the number of the line that TeX would show on an error message if an error occurred now. (This parameter and `\badness` are "read only" in the same way as `\lastpenalty`: You can use them in the context of a ⟨number⟩, e.g., by saying '`\ifnum\inputlineno>\badness ... \fi`' or '`\the\inputlineno`', but you cannot set them to new values.)

10. Not looking at error context

There's a new integer parameter called `\errorcontextlines` that specifies the maximum number of two-line pairs of context displayed with TeX's error messages (in addition to the top and bottom lines, which always appear). Plain TeX now sets `\errorcontextlines=5`, but higher level format packages might prefer `\errorcontextlines=1` or even `\errorcontextlines=0`. In the latter case, an error that previously involved three or more pairs of context would now appear as follows:

```
! Error.
⟨somewhere⟩ The \top
                        line
...
l.123 \The
                bottom line.
```

(If \errorcontextlines<0 you wouldn't even see the '...' here.)

11. Output recycling

One more new integer parameter completes the set. If \holdinginserts>0 when TEX is putting the current page into \box255 for the \output routine, TEX will not move anything from insertion nodes into the corresponding boxes; all insertion nodes will stay in place. Designers of output routines can use this when they want to put the contents of box 255 back into the current page to be re-broken (because they might want to change \vsize or something).

12. Exceptions to upward compatibility

The new features of TEX and METAFONT imply that a few things work differently than before. I will try to list all such cases here (except when the previous behavior was erroneous due to a bug in TEX or METAFONT). I don't know of any cases where users will actually be affected, because all of these exceptions are pretty esoteric.

• TEX used to convert the character strings ^^0, ^^1, ..., ^^9, ^^a, ^^b, ^^c, ^^d, ^^e, ^^f into the respective single characters p, q, ..., y, !, ", #, $, %, &. It will no longer do this if the following character is one of the characters 0123456789abcdef.

• TEX used to insert no character at the end of an input line if \endlinechar>127. It will now insert a character unless \endlinechar>255. (As previously, \endlinechar<0 suppresses the end-of-line character. This character is normally 13 = ASCII control-M = carriage return.)

• Some diagnostic messages from TEX used to have the notation ["80] ... ["FF] when referring to characters 128...255 (for example when displaying the contents of an overfull box involving fonts that include such characters). The notation ^^80 ... ^^ff is now used instead.

• The expressions char128 and char0 used to be equivalent in METAFONT; now **char** is defined modulo 256 instead. Hence char-1 = char255, etc.

• INITEX used to forget all previous hyphenation patterns each time you specified \patterns. Now all hyphenation pattern specifications are cumulative, and you are not permitted to use \patterns after a paragraph has been hyphenated by INITEX.

• TEX used to act a bit differently when you tried to typeset missing characters of a font. A missing character is now considered to be a word boundary, so you will get slightly more diagnostic output when \tracingcommands>0.

• TEX and METAFONT will report different statistics at the end of a run because they now have a different number of primitives.

• Programs that use the string pool feature of TANGLE will no longer run without changes, because the new TANGLE starts numbering multicharacter strings at 256 instead of 128.

• INITEX programs must now set \lefthyphenmin=2 and \righthyphenmin=3 in order to reproduce their previous behavior.

◇ Donald E. Knuth
Department of Computer Science
Stanford University
Stanford, CA 94305

PubliC METAFONT Available

Editor's note: Klaus Thull announces that, as of 6 October, PubliC METAFONT is available. PubliC METAFONT compiles with Turbo Pascal v.4 or 5 and has passed the trap test. As with its companion, PubliC TEX (see *TUGboat* 10#1, pp. 15–22), this program has virtual memory (and is also somewhat slow).

Work is going on at sites other than Klaus' for improving performance and video, as has been the case with PubliC TEX. Distribution is now being handled by DANTE, the German speaking TEX users association. The changefile for version 0 of PubliC METAFONT is available at

Bitnet: listserv@dhdurz1