Guidelines for creating portable METAFONT code

Don Hosek

The TEX community is currently starved for new public domain meta-fonts and chances are that no matter how useless you think that some meta-font you may have created might be, there are at least forty people "out there" who would want it.

So with this the situation and chances that your code might find its way to operating systems vastly different than your own, I would like to offer the following guidelines to METAFONT designers for ensuring that their code can be run on other systems with a minimum of effort.

1 Internal documentation

METAFONT sources can go a long way and be transmitted in many forms. Just because you might send your source out in some encapsulated format (*e.g.,* `tar` or `arc`) doesn't mean that they will always be redistributed as such. More than once I've found myself with a file with an ambiguous name like `newzm.mf` and had no indication what it was part of or intended for. I would recommend that each source file you create contain the following information:

- The name of the file (this often can get lost).
- The last revision date. If you are modifying an existing METAFONT file, you should retain the old file's revision date and add your own along with a description of all changes made. This will allow easy updates to your file if the original file is later revised.
- The name of the package that the file is part of (*e.g.,* "CM Pica", "Pandora", etc.).
- Your name. This will make it easier for later users to track you down for complaints/suggestions/whatever. You may also want to include your current institutional affiliation and e-mail address as well.
- A brief description of the purpose of the file. This will make things easier on the individual who later attempts to follow the logic of your code.

None of these are *necessary* for allowing a METAFONT file to run other systems, but they will serve as an aid to users on other systems attempting to install your font.

Pierre MacKay has suggested the scheme shown in Figure 1 for this internal documentation (based on the file comments used by J. E. Pittman).

2 File names

The most important consideration when selecting filenames is to do your best to avoid file name conflicts. METAFONT's rule for selecting a file is to look for the indicated file in the current directory, then to look in `MFINPUTS` for the file. Personally, I believe that the current directory when METAFONT is run should be the one on which the METAFONT output will ultimately end up, so in that case, we are left with essentially a flat file space.[1] Thus it is essential to try not to have file name conflicts.

While in theory, this is a nice principle, it might be asked, "I can easily check my names against fonts that I have, but how can I be sure that I won't conflict with some odd font from someone else?" The short answer is that you can't. The longer answer is that it's possible to reduce the probability, simply by having all the files begin with the same initial combination of letters. For example, in an extra symbols font that I'm developing for use with internal fonts in the Xerox laser printers here on campus, I prefix each file used with the letters "cs" (for Century Schoolbook). While I can't know for certain that I've avoided all conflicts in this manner, chances are that name conflicts will not occur. As an added guard against conflicts, you might want to pick an additional arbitrary letter and tack it onto the file name to further guard against name conflicts (personally, I'm partial to "q").

Another important consideration is the fact that all IBM systems (including PC's) have a filename restriction of eight characters. Now, it's not strictly necessary to make all file names eight characters or less, but it would be helpful to at least guarantee that file names are unique to the first eight characters. The PC restriction makes this especially important as PC floppies are a convenient, inexpensive, and almost universally readable format for exchanging information. In addition, in a recent survey of TEX users in TEXMAG, over half the respondents used TEX on an IBM PC or compatible. Ignoring the eight character restriction can make your font inaccessible to a significant portion of the TEX community.

[1] Some operating systems, like IBM's VM/CMS give you a flat file space whether you want it or not; while one might be tempted to simply choose to ignore CMS as a METAFONT operating system, this is not feasible since the speed of IBM mainframes makes running METAFONT under CMS quite desirable—running METAFONT on Computer Modern fonts on an IBM 3081 took an average of 30 seconds per font!

```
% File:       MF Inputs U_Wash.mf
% Author:     Pierre A. MacKay
% Internet:   mackay@cs.washington.edu
% Bitnet:     mackay@cs.washington.edu
% Date:       November 27, 1988
%
% This is the University of Washington collection of |mode_def|s
% together with the macros to provide font-wide specials describing the
% |mode_def| that is used for each generated font, and the Xerox-world
% comments in the tfm file.  If a '?' is typed as the first response
% to the '*' prompt after this or a derived base file is loaded,
% a list of all current |mode_def|s will be given.
%
% This file follows a convention that has emerged in the discussion
% of |mode_def|s in TUGboat.
%    1.  The print engine is identified wherever possible, rather than
%        the printer which incorporates that print-engine.
%    2.  Because |mode_def| names may not contain digits, each digit is
%        given its full name, as in RicohFourZeroEightZero.
%
% WARNING: Some of the modes have never actually been tested
```

Figure 1: A model for internal METAFONT documentation

If you use a non-standard extension for any of your METAFONT files, you should take care that it is three characters or less, for the same reasons as the eight character limit above.

A file name should ideally consist only of the letters a–z and the digits 0–9. The first character of the name or extension should be a letter (some operating systems choke on file names beginning with numerals). When specifying a file name on an **input** statement, use all lower case; this will make life easier for the Unix people.

Finally, *never* include an explicit directory path on a METAFONT **input** statement. Since area names are necessarily system dependent, this guarantees that your code will not be portable.

3 MFT compatibility

MFT is a system for producing "pretty-printed" listings from METAFONT files; it was used in the production of Volume E of *Computers and Typesetting* and portions of *The METAFONTbook*. A complete description of MFT's capabilities and conventions is beyond the scope of this article,[2] but there are some simple things you can do to prevent MFT from blowing up.

[2] An MFT manual is in the works

- Use only a single percent sign on comments. MFT uses multiple percent signs to flag special handling code.
- Make sure that all comments are valid TeX input. If you refer to any METAFONT commands, variables, etc. enclose them in |...|.
- If you *must* comment out lines of code, either enclose the entire line in |...| as noted above or use four percent signs (%%%%) to comment out the line.

4 Coding considerations

Most of the METAFONT code you write will be portable by default, but there are a few things that should always be taken into consideration:

- Never set *mode* inside the file. The proper way to invoke METAFONT is to say

  MF \mode=*whatever*; input *file*

  This eliminates the need to specify *mode* inside the file. Similarly, *mag* should also not be specified in a METAFONT file.
- Keep the parameter definitions in a separate file from character definitions. While you might need a given font only at, say ten point roman, it's possible that someone else might want a nine point boldface of the characters you've designed. If you follow the model of Knuth's Computer Modern, this sort of modification will be much easier.

4.1 Specifying dimensions

Always use sharped units when defining a dimension in your code and convert the sharped unit to pixels using one of the METAFONT commands listed on p. 268 of *The METAFONTbook*. I have encountered fonts which have specified things like **pickup pencircle** scaled 2 which will work fine on a dot matrix printer or even a write-black laser printer but looks awful on a write-white laser printer. What should have been done instead would be to specify some dimension such as *tiny*# which would later be converted using **define_blacker_pixels** into the appropriate pixel value for the output device.

METAFONT's sharped units provide a method for specifying units in a device-independent way. Rather than specify the widths of lines and other dimensions in terms of pixels, one first specifies units in terms of sharped units (you are given all of TeX's dimensions to begin with), then converts them with one of the macros listed below. Each is called in the form **define_pixels**(*var_one, var_two*) where there can be as many variable names listed between the parentheses as necessary. For each variable name given, METAFONT will set its value according to a conversion into pixels from the corresponding sharped variables. In the example above, *var_one* and *var_two* would hold the pixel values of *var_one*# and *var_two*#.

define_pixels Converts a sharped variable into pixels. This is done through a simple conversion. This should be used for variables which would not need any of the corrections described below. For example, a parameter used in calculating the widths of characters (such as Computer Modern's *u*#) would be converted into pixels with this command.

define_whole_pixels Converts a sharped variable into an integral number of pixels. This is normally used for variables which indicate the placement of certain points in the character.[3]

define_whole_vertical_pixels Converts a sharped variable into an integral number of vertical pixels. This is used for the same sort of variables as **define_whole_pixels**, but takes into account any non-unit aspect ratio which may be used for the output device. This is generally used for vertical positioning while **define_whole_pixels** is used for horizontal postioning.

[3] For a complete discussion of why using integral values for various parameters is important, see Chapter 24 of the *The METAFONTbook*.

define_good_x_pixels Converts a sharped variable into a value such that a pen drawn using the value as an x-coördinate will have its left edge on a pixel boundary. You must have a current pen selected for this to work. This is generally used for character sets (such as the one used in the METAFONT logo) where many of the characters are drawn using a single pen.

define_good_y_pixels Converts a sharped variable into a value such that a pen drawn using the value as the y-coördinate will have its top edge on a pixel boundary. This is the vertical analogue to **define_good_x_pixels**.

define_blacker_pixels Converts a sharped variable into pixels adding METAFONT's *blacker* to the value obtained. This should be used for a variable which will determine the width of lines drawn or pens used. This is a very important definition since without it, METAFONT's **mode_def** convention is almost useless.

define_whole_blacker_pixels Converts a sharped variable into an integral number of pixels taking METAFONT's *blacker* into account. This should be used for variables which will determine the width of lines drawn or pens used which should be set to an integral value.

define_whole_vertical_blacker_pixels Converts a sharped variable into an integral number of vertical pixels. This has the same relationship to **define_whole_blacker_pixels** as **define_whole_vertical_pixels** has to **define_whole_pixels**.

define_corrected_pixels Converts a sharped variable into a pixel value after taking into account the curve overshoot parameter (METAFONT's *o_correction*). This should be used on variables which give the overshoot for a curved portion of a character (*e.g.,* the bottom of "U"). *The METAFONTbook* has details on when this is appropriate.

define_horizontal_corrected_pixels Similar to **define_corrected_pixels** but does the rounding for a horizontal value.

The METAFONT logo font (which is described throughout *The METAFONTbook* and listed in its entirety in Appendix E of that work) is a good simple example to see how these different METAFONT commands are used.

4.2 Compatibility with Computer Modern

Unless your font is designed explicitly for use with some non-Computer Modern font (*e.g.,* extra symbols for use of TeX with a printer-resident font), it is probably a good idea to plan your type so that it is visually compatible with Computer Modern. You will probably also want to follow the existing TEX coding schemes (except for odd fonts such as an astronomical symbols font) as well. These practices carry with them several benefits:

- By following existing coding schemes you make it easier to achieve compatibility with existing TEX macros.
- Visual compatibility with Computer Modern allows you to use CM fonts for things such as typewriter type and math if you so choose.

  In addition, if type "A" is visually compatible with Computer Modern and type "B" is visually compatible with Computer Modern then types "A" and "B" should be visually compatible with each other.

The primary objective when striving for "visual compatibility" is to guarantee that the characters should align well with Computer Modern. At the very least, baselines of characters should match well. To allow use of Computer Modern math fonts with your typeface, the weights of the characters should roughly correspond to the weights of corresponding characters in CM.

As an example, consider Figure 2 which mixes Computer Modern and Concrete together in several contexts. These two typefaces have a roughly corresponding character grid, but the difference in weights produces an odd mixture when the two are combined.[4] Overall, the samples above give some indication of the flexibility obtained by striving for compatibility with Computer Modern.

Concrete and Computer Modern Roman will not mix well.
Concrete and `Computer Modern typewriter type will` blend somewhat better.
Concrete does not produce optimum results with $\Gamma + k\alpha = 0$ Computer Modern math.

Figure 2: Mixing Concrete and Computer Modern in some different contexts.

◇ Don Hosek
3918 Elmwood
Stickney, IL 60402
Bitnet: `u33297@uicvm`

[4] In fact, as was explained in *TUGboat* **10**(1), these fonts were designed for use with the Euler math fonts.