- On an IBM 370/3033 with Pascal/VS at Stanford CIT (Eagle Berns).
- On a VAX (VMS) at Oregon Software (Barry Smith).
- On an IBM 370/3022 (VM-CMS) with SLAC-Pascal at the University of Pisa (Gianfranco Prini). They printed the DVI files on a Versatec.
- On a Univac 1100/82 at the University of Wisconsin (Ralph Stromquist). Output is to a Compugraphic 8600. (See report, p. 51.)

From the information sent to Stanford, we gather that the Pascal compilers being employed in the installations of TEX are:

IBM 370: Pascal-VS, SLAC-Pascal, Pascal-8000
UNIVAC: U. of Wisconsin Pascal, Pascal-8000
PDP-10: Hamburg Pascal
VAX: (See report by Janet Incerpi, p. 49.)

Note: Charles Lawson (Jet Propulsion Lab.—Caltech) has produced two short reports that can help in reprogramming the SYSDEP module of TEX-Pascal. (Both are reprinted in this issue, pp. 20 and 32.)

* * * * * * * * * * *

TEX FONT METRIC FILES

— or —

What happens when you say "\font A=CMR10"

David Fuchs

When you tell TEX that you will be using a particular font, it has to find out information about that font. It gets this information from what are known as .TFM files. For instance, when you say `\font A=CMR10` to TEX (`\:A=CMR10` in the old TEX lingo), TEX looks around for a file called CMR10.TFM, and reads it in. If CMR10.TFM is not to be found, TEX will give you the error message `"Lookup failed on file CMR10.TFM"`, and you will be out of luck as far as using CMR10 is concerned.

What does TEX want with the TFM files? Generally speaking, a font's TFM file contains information about the height, width and depth of all the characters in the font, plus kerning and ligature information. So, CMR10.TFM might say that the lower-case "d" in CMR10 is 5.55 points wide, 6.94 points high, etc. This is the information that TEX uses to make its lowest-level boxes—those around characters. See the TEX manual (p. 41) for information about what TEX does with these boxes. Note that TFM files do NOT contain any device-dependent description of the font (such as the raster description of the characters at a certain resolution). Remember that the program TEX does not deal with pixels. Only device-drivers that read TEX's DVI output files use that sort of information.

Where do .TFM files come from? The best way to get a TFM file is with **METAFONT**. Font designers should include the **METAFONT** instructions that specify the width, height, etc. of each character they design. The **METAFONT** manual contains details and examples of how to do this—see the index entries for **charwd**, **charht**, **chardp**, etc. If this is done, then when **METAFONT** is run on CMR10, it produces CMR10.TFM. (Depending on what "mode" it is run in, it also makes CMR10.FNT, CMR10.ANT, CMR10.VNT, or CMR10.OC. These are all different formats for files containing the raster description of the font. Drivers for various devices require one or another of these files.)

Whatever happened to the TFX format that the TEX and **METAFONT** manuals actually refer to? Is this just a misprint for TFM? No—TFM files take the place of TFX files. The differences are conceptually small; they both contain more or less the same information. The main reason for changing TEX and **METAFONT** from using/making TFX files to TFM files is that TFX files were based on 36-bit words. This proved to be a real problem for people running Pascal TEX, especially on 32-bit machines. The format of TFM files assumes 8-bit bytes, packed four to a 32- or 36-bit word. They are readily adapted for use on 16-bit machines, too. While the format was being changed, a few new bits of information were added, too.

What if I have fonts that I want TEX to know about that were not made with **METAFONT**? Don't despair—we have two programs, TFTOPL and PLTOTF, that convert TFM files to readable, editable format, and back again. For instance, if we run TFTOPL on CMR10.TFM, it makes CMR10.PL, an excerpt of which follows ("R" means a floating-point number is coming up (all dimensions given in this form are in terms of the DESIGNSIZE); "C" means that a character is next; "O" means an octal value for a character that isn't an ASCII printing character is next):

```
(FAMILY CMR)
(DESIGNSIZE R 10.0000000)
(CODINGSCHEME TEX TEXT)
(TEXINFO
        (SPACE R .3333330)
        (STRETCH R .1666670)
        (SHRINK R .1111107)
        (XHEIGHT R .4444447)
        (QUAD R 1.0000000)
        (EXTRASPACE R .1111107)
        )
```

```
(LIGTABLE
        (LABEL C f)
        (LIG C 1 O 174)
        (LIG C f O 173)
        (LIG C 1 O 175)
        (LIG C t O 44)
        (STOP)
        (LABEL C o)
        (KRN C o R  .0277777)
        (KRN C x R -.0277777)
        (STOP)
        )
(CHARACTER C d
        (CHARWD R  .5555553)
        (CHARHT R  .6944447)
        (CHARDP R  .0000000)
        )
```

Which says: This font is in the CMR family, its size is 10.0 points, and it's a regular TEX font for text. When TEX uses this font, it should know that the glue between words should be 3.33pt plus 1.66pt minus 1.11pt. TEX will also know that the x-height of CMR10 is 4.44 pt, a "quad" of space is 10 pt, and the extra amount of space at the end of a sentence is 1.11 pt. (see the **METAFONT** (Appendix F) and TEX (everywhere) manuals for more on these parameters). Also, TEX should recognize that whenever "f" is followed by "i", "f", "l", or "t", that it's time for a ligature: an occurrence of "f1" should be replaced by the character in octal position 174 in CMR10 ("fl"; see the Appendix of the TEX manual that shows font tables to verify this), etc. Whenever "o" is next to another "o", they should be moved apart by .277 pt more then they would otherwise be (based on their widths), but an "o" should be moved .277 pt closer to a following "x". Finally, the character "d" has width 5.55 pt, height 6.94 pt, and depth 0 pt. The full CMR10.PL file has lots more in its `LIGTABLE` and many more `CHARACTER` descriptions, of course.

If we changed the line

```
        (CHARWD R  .5555553)
```

to

```
        (CHARWD R  .700)
```

and ran PLTOTF, making a new CMR10.TFM, TEX would now think that "d" had width 7 pt in CMR10.

So, if you have your own fonts that you'd like TEX to know about, just make PL files for them, and then run PLTOTF to make TFM files. With luck, your fonts will be in some computer readable form such that the PL files can be made with a fairly simple program. Note that it is perfectly legal to have a TFM (or PL) file that specifies no kerns or ligatures.

In fact, TFM files may contain a fair amount more information about a font than just the heights, widths, depths, kerns and ligatures. Most of these extra paramaters only come up in the special math fonts. The **METAFONT** (Appendix F again) and TEX manuals talk about these parameters in more detail. Below is a complete but scary description of all the bits in TFM files. This description was actually excerpted from a comment in the SAIL-language version of TEX.

* * * * * * * * * * * *

This definition of TFM files is due to Lyle Ramshaw.

Each font used by TEX has an associated font information file. The name of this file is obtained by appending the extension code ".TFM" to the font file name. For example, the TEX font metrics for the font CMR10 appear on the file CMR10.TFM. These .TFM files are written with 32 bits in each word, to facilitate their transportability. When they sit in the file systems of 36-bit machines, these 32 data bits will be left-justified in the 36-bit word, leaving the rightmost four bits zero.

The first 6 words of the .TFM file contain twelve 16-bit integers that give the lengths of the various portions of the file, packed two to a 32-bit word. These twelve integers are, in order:

- `lf` = length of entire file in words,
- `lh` = length of header data,
- `bc` = first character code in font,
- `ec` = last character code in font,
- `nw` = number of words in width table,
- `nh` = number of words in height table,
- `nd` = number of words in depth table,
- `ni` = number of words in italic correction table,
- `nl` = number of words of lig/kern program,
- `nk` = number of words in kern table,
- `ne` = number of words in extensible character table,
- `np` = number of font parameters.

In .TFM format, the subfields of a word are always allocated in left-to-right (BigEndian) order. Thus, the first two integers in this list, `lf` and `lh`, are packed into the first word of the .TFM file with `lf` on the left and `lh` on the right.

These lengths are not all independent: they must obey the relation

`lf=6+lh+(ec-bc+1)+nw+nh+nd+ni+nk+nl+ne+np.`

The rest of the .TFM file is a sequence of ten data arrays as specified below:

```
HEADER= ARRAY[0:lh-1] of Stuff
FINFO= ARRAY[bc:ec] of FInfoEntry
WIDTH= ARRAY[0:nw-1] of FIX
HEIGHT= ARRAY[0:nh-1] of FIX
DEPTH= ARRAY[0:nd-1] of FIX
CHARIC= ARRAY[0:ni-1] of FIX
LIG/KERN= ARRAY[0:nl-1] of LigKernStep
KERN= ARRAY[0:nk-1] of FIX
EXT= ARRAY[0:ne-1] of ExtRecipe
PARAMS= ARRAY[1:np] of FIX.
```

A FIX is a one-word representation of a real number in a fixed-point fashion. FIXes are used in .TFM format to enhance transportability. A FIX is a signed quantity, with the two's complement of the entire FIX used to represent negation. Of the 32 bits in the word, 12 are to the left and 20 are to the right of the binary point. This means that a FIX has 1 bit of sign, 11 bits of integer, and 20 bits of fraction. Note that this limits the size of real numbers that a FIX can represent: the largest FIX is roughly 2000.

The first data array is a block of header information, general information about the font. Currently, the header contains 18 words, allocated as described below. In the future, new fields might be added at the end of the header block.

```
HEADER=
        [
        CheckSum: 1 word
        DesignSize: FIX (1 word)
        CharacterCodingScheme: 10 words
        ParcFontIdentifier: 5 words
        Random word (=Header[17]
                        is broken up as follows:)=
                [
                SevenBitSafe: 1 bit
                unusedspace: 23 bits
                ParcFaceByte: 8 bits
                ]
        ]
```

The CheckSum field is used to hold a unique identifier of some sort that describes this version of the font. This unique ID is put by **METAFONT** into both the rasters and metrics. TEX finds it in the metrics, and stores it in the .DVI file. Thus, a spooler can check the unique ID in the .DVI with the unique ID in its rasters, to provide a guarantee that TEX was working with metric data for the current rasters. **METAFONT** computes this checksum from the metric information in the .TFM file.

The DesignSize of the font is the size that the font was intended to look good at, or, to put it another way, the nominal size of the font when it is printed at a magnification of 1.0. For unusual fonts such as CMDUNH and CMATHX, the DesignSize is more-or-less arbitrary. The DesignSize is stored as a FIX with the units "points".

The CharacterCodingScheme field is supposed to specify what the character code to symbol translation scheme is in this font. The coding scheme is stored in 10 words=40 bytes of the .TFM file, as a string. The first byte gives the length of the string, the next n bytes are the characters, and the last $(39 - n)$ bytes are zeros (where "first" and "next" imply working from left-to-right). Some common coding scheme names are:

CharacterCodingSchemes:

TEX TEXT	
TEX TYPEWRITER TEXT	
TEX MATHIT	
TEX MATHSY	
TEX MATHEX	
UNSPECIFIED	— default, means no information
GRAPHIC	— special purpose code, non-alphabetic
ALPHABETIC	— means alphabet agrees with ASCII at least
ASCII	— means exactly ASCII
PARC TEXT	— Times Roman and Helvetica, for example
SUAI	
CMU	
MIT	

Fonts are universally referred to by their file names: for example "CMR10" for Computer Modern Roman 10 point, "CMTT" for Computer Modern TypeWriter Type 10 point, etc. The TEX user specifies the font by giving this string name, the TEX output module finds the metric file by using this name with the extension .TFM, and the various printers store the rasters as files with this name and some other extension.

TEX can only handle character codes that are seven bits in length. But the .TFM format always allows a full eight bits for a character code, so the high-order bit of all characters specified must be zero.

The HEADER data is followed by the FINFO table, which is an array of FInfoEntrys. This array is indexed from bc to ec, and hence contains $(ec - bc + 1)$ entries. Each FInfoEntry is one word in length. An FInfoEntry is a compacted structure with the following format (fields allocated from left to right once again):

```
FInfoEntry:
        [
        WidthIndex: 8 bits
        HeightIndex: 4 bits
        DepthIndex: 4 bits
        CharIcIndex: 6 bits
        TagField: 2 bits
        Remainder: 8 bits
        ]
```

The fields in the FInfoEntry do not give the character width, height, etc. directly, they are indices into secondary tables. Thus, up to 256 different widths may appear among the 256 characters of a single font, and up to 16 different heights, 16 different depths, and 64 different italic corrections. The actual widths, heights, depths and italic corrections are stored in the .TFM file as arrays of FIXes with "em" as their units. TEX reads in these FIXes, converts them to floating point form, scales them by multiplying by the desired font size (in points), and stores them into internal character metric arrays.

The CharIc field is used both for the italic correction of ordinary characters and for mathop kerns. In particular, for mathops such as summation and integral signs, the CharIc field points to a "kern" which, if nonzero, means that limits are normally set to the right and the lower limit is shifted left by this kern value. If the kern is 0, limits in display style will be centered above and below the operator. (To change between centering and attaching at the right, one writes "\limitswitch" after the operator.)

A note on non-existent characters: all character codes outside of the range [bc,ec] represent characters that do not exist in the font. Any codes in the range [bc,ec] that represent non-existent characters will have their FInfoEntrys identically equal to 0. The WIDTH, HEIGHT, DEPTH, and CharIc arrays will each be guaranteed to have a FIX of 0.0 in their 0'th position. Thus, failing to notice that a character is non-existent won't lead a program to use irrelevant metric data for that character code. Furthermore, any characters that really do exist in the font will be guaranteed to have a WidthIndex that is nonzero. Thus, a character is non-existent iff its WidthIndex is zero, and also iff its entire FInfoEntry is zero. If there are any actual characters in the font whose width just happens to be precisely zero, the WIDTH array will contain two zero FIXes: one at index 0, which is used for all of the non-existent characters, and one somewhere else.

The remaining portion of the FInfoEntry is used for several different purposes, depending upon the value of the tag field. The TagField portion of the FInfoEntry has one of four values:

tag=0 ⇒ this is a vanilla character, Remainder is unused.

tag=1 ⇒ character has a ligature-kerning program: the Remainder field is the index in the LIG/KERN array of the first step of the program.

tag=2 ⇒ character is part of a chain of characters of ascending sizes ("charlist"): the Remainder field gives the character code of the next larger character in the chain.

tag=3 ⇒ character code represents an extensible character, one that is built up out of smaller pieces and can be made arbitrarily large: the Remainder field is an index into the EXT array. The ExtRecipe at that position in the EXT array describes what the pieces are.

(The taglist and tagvar options are usually used only in math extension fonts.)

The LIG/KERN array is a program in a simple programming language that gives instructions about what to do for special letter pairs. Each step in this program occupies one word:

```
LigKernStep:
        [
        StopBit: 1 bit
            # means this is a final program step
        unusedspace: 7 bits
        NextChar: 8 bits
            # if this is the next character, then...
        TagBit: 1 bit
        unusedspace: 7 bits
        Remainder: 8 bits
        ]
```

If the TagBit is 0, this step in the program describes a ligature. In that case, the Remainder consists of the character code of the ligature that should be substituted for the current character pair. If the TagBit is 1, this step describes a kern, and the Remainder field is an index into the KERN array. The KERN array is simply an array of FIXes, pure numbers that should be scaled to give distances in the same way as the elements of the WIDTH, HEIGHT, DEPTH, and CharIc arrays.

An ExtRecipe is a one-word quantity that should be viewed as four bytes (allocated left-to-right, of course):

```
ExtRecipe:
        [
        top: byte
        mid: byte
        bot: byte
        ext: byte
        ]
```

The height and width fields in the FInfoEntry of the extensible character give the metrics of the component, not of the built-up symbol itself, since the built-up symbol will have variable size. If top, middle, or bottom portions are zero, the extension component runs all the way through that portion of the symbol, otherwise it directly abuts these portions. The built-up symbol is formed by including an integral number of extension components. If there is a middle, the same number of extension components will appear above and below. For example, a left brace has all four components specified, while a double || (the cardinality or norm symbol) has only an extension part. The floor and ceiling brackets are like regular brackets, but without top or bottom, respectively. The width of the extension component is assumed to be the width of the entire built-up symbol. If any byte is 0, it indicates that the corresponding piece of the extensible character does not exist. Otherwise, the contents of the byte is the character code of the piece: top, middle, bottom, or extender respectively.

The rest of the .TFM file is the PARAMS array, a table of font parameters that are used by TEX, stored as FIXes. All of these parameters are distances except for the first one, "slant": hence all except for "slant" should be scaled by the font size by TEX when being read in from the .TFM file. Since slant is a pure number, it should not be scaled. [The following table of parameters is printed in clearer form on pages 98–100 of the **METAFONT** manual.]

slant the amount of italic slant (e.g. slant=.25 means that when going up one unit, go .25 units to the right—this is used in placing accents over characters)

space a real number that says how wide blank spaces are (Note that TEX doesn't use character number '40 for spaces, that character can be non-blank in the font)

spacestretch
the stretch component of the glue for spacing

spaceshrink
the shrink component of the glue for spacing

xheight the height of lowercase "x" (default positioning for accents)

quad the width of one "em"

extraspace
the amount added to space after periods (and in general when the spacefactor is greater than 2)

Mathematics fonts used as \mathsy and \mathex contain important additional parameter information. In a \mathsy font, the extra parameters start right after "quad", that is, there is no "extraspace" parameter. The \mathsy parameters are

mathspace
if nonzero, the amount of space that will be used for all nonzero space in math formulas (for fixed-width output)

num1,num2,num3
amount to raise baseline of numerators in display or nondisplay or nondisplay-atop styles, respectively

denom1,denom2
amount to lower baseline of denominators

sup1,sup2,sup3
amount to raise baseline of superscripts if
1) display style
2) nondisplay nonvariant style
3) variant style

sub1,sub2
amount to lower baseline of subscripts if superscript is
1) absent
2) present

supdrop,subdrop
amount below top or bottom of large box to place baseline if the box has a superscript or subscript in this size

delim1,delim2
size of \comb delimiters in
1) display
2) nondisplay style

axisheight
height of fraction lines above the baseline (this is midway between the two bars of = sign)

A \mathex font includes the first seven standard parameters (including extraspace), and then has six parameters used to govern formula setting:

defaultrulethickness
the thickness of \over and \overline bars

bigopspacing(1),(2)
the minimum glue space above and below a large displayed operator, respectively

bigopspacing(3),(4)
the minimum distance between a limit's baseline and a large displayed operator, when the limit is above, below

bigopspacing(5)
the extra glue placed above and below displayed limits