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 8800. (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 Incerni, 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 \texttt{\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 \texttt{\font A=CMR10} to \TeX\ (\texttt{\font 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 TFM'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 METAFTER. Font designers should include the METAFTER instructions that specify the width, height, etc. of each character they design. The METAFTER 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 METAFTER 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 METAFTER 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 METAFTER from using/making TFX files to TFM files is that TFX files were based on 32-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 64-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 METAFTER? Don't despair—we have two programs, TFTPOLL and PLTOTF, that convert TFM files to readable, editable format, and back again. For instance, if we run TFTPOLL 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 \SIZES); "C" means that a character is next; "T" means an octal value for a character that isn't an ASCII printing character is next):}
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 parameters only come up in the special math fonts. The TeX (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:

1. length of entire file in words,
2. length of header data,
3. first character code in font,
4. last character code in font,
5. number of words in width table,
6. number of words in height table,
7. number of words in depth table,
8. number of words in italic correction table,
9. number of words of lig/kern program,
10. number of words in kern table,
11. number of words in extensible character table,
12. 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, 1f and 1h, are packed into the first word of the .TFM file with 1f on the left and 1h on the right.

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

\[ 1f = 6 + 1h + (ec-bc+1) + nw + nh + nd + ni + nk + nl + ne + np. \]
The rest of the .TFM file is a sequence of ten data arrays as specified below:

- HEADER= ARRAY[0:1h-1] of Stuff
- FrmX= ARRAY[bc:sc] of FrmInfoEntry
- WIDTH= ARRAY[0:nw-1] of FIX
- HEIGHT= ARRAY[0:nh-1] of FIX
- DEPTH= ARRAY[0:nd-1] of FIX
- CHARIC= ARRAY[0:n1-1] of FIX
- LIG/KERN= ARRAY[0:n1-1] 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 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
   unused: 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. TpX 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 TpX 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

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 TpX user specifies the font by giving this string name, the TpX 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.

TpX 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 FrmInfoEntries. This array is indexed from bc to sc, and hence contains (sc — bc + 1) entries. Each FrmInfoEntry is one word in length. An FrmInfoEntry 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 \texttt{\textbackslash limitsvitch}.

A note on non-existent characters: all character codes outside of the range \texttt{[bc, ec]} represent characters that do not exist in the font. Any codes in the range \texttt{[bc, ec]} that represent non-existent characters will have their FInfoEntry 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 if 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:

- \texttt{tag=0} ⇒ this is a vanilla character, Remainder is unused.
- \texttt{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.
- \texttt{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.
- \texttt{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 \texttt{taglist} and \texttt{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, \texttt{slant}: hence all except for \texttt{slant} should be scaled by the font size by TeX when being read in from the .TFM file. Since \texttt{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.]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{slant}</td>
<td>the amount of italic slant (e.g. \texttt{slant}=.25 means that when going up one unit, go .25 units to the right—this is used in placing accents over characters)</td>
</tr>
<tr>
<td>\texttt{space}</td>
<td>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)</td>
</tr>
<tr>
<td>\texttt{spaceshrink}</td>
<td>the shrink component of the glue for spacing</td>
</tr>
<tr>
<td>\texttt{xheight}</td>
<td>the height of lowercase &quot;x&quot; (default positioning for accents)</td>
</tr>
<tr>
<td>\texttt{quad}</td>
<td>the width of one &quot;em&quot;</td>
</tr>
<tr>
<td>\texttt{extraspace}</td>
<td>the amount added to space after periods (and in general when the spacefactor is greater than 2)</td>
</tr>
</tbody>
</table>

Mathematics fonts used as \texttt{\mathbf{math}} and \texttt{\textit{math}} contain important additional parameter information. In a \texttt{\mathbf{math}} font, the extra parameters start right after “quad”, that is, there is no \texttt{"extraspace"} parameter. The \texttt{\mathbf{math}} parameters are

- \texttt{mathspace} if nonzero, the amount of space that will be used for all nonzero space in math formulas (for fixed-width output)
- \texttt{num1,num2,num3} amount to raise baseline of numerators in display or nondisplay or nondisplay-stop styles, respectively
- \texttt{denom1,denom2} amount to lower baseline of denominators
- \texttt{sup1, sup2, sup3} amount to raise baseline of superscripts if 1) display style 2) nondisplay nonvariant style 3) variant style
- \texttt{sub1, sub2} amount to lower baseline of subscripts if superscript is 1) absent 2) present
- \texttt{supdrop, subdrop} amount below top or bottom of large box to place baseline if the box has a superscript or subscript in this size
- \texttt{delim1, delim2} size of \texttt{\textbackslash{}comb} delimiters in 1) display 2) nondisplay style
- \texttt{axisheight} height of fraction lines above the baseline (this is midway between the two bars of \texttt{\textbackslash{}approx} sign)

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

- \texttt{defaultrulethickness} the thickness of \texttt{\textbackslash{}over} and \texttt{\textbackslash{}overline} bars
- \texttt{bigopspace(1), (2)} the minimum glue space above and below a large displayed operator, respectively
- \texttt{bigopspace(3), (4)} the minimum distance between a limit’s baseline and a large displayed operator, when the limit is above, below
- \texttt{bigopspace(5)} the extra glue placed above and below displayed limits