1. Introduction.

2. Needed headers and macros

```
#define qi(A) (quarterword)(A) /* to store eight bits in a quarterword */
#define null_font 0 /* the font_number for an empty font */
#define false 0
#define true 1
#define hlp1(A) mp->help_line[0]=A; }
#define hlp2(A,B) mp->help_line[1]=A; hlp1(B)
#define hlp3(A,B,C) mp->help_line[2]=A; hlp2(B,C)
#define help3 { mp->help_ptr=3; hlp3 /* use this with three help lines */
#include <w2c/config.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mplib.h"
#include "mpmp.h" /* internal header */
#include "mpmath.h" /* internal header */
#include "mpstrings.h" /* internal header */
⟨ Declarations 3 ⟩;
⟨ Preprocessor definitions ⟩
```

3. The *font_ps_name* for a built-in font should be what PostScript expects. A preliminary name is obtained here from the TFM name as given in the *fname* argument. This gets updated later from an external table if necessary.

```
⟨ Declarations 3 ⟩ ≡
  font_number mp_read_font_info(MP mp, char *fname);
```

This code is used in section 2.

4. *font_number mp_read_font_info*(MP *mp*, **char** ∗*fname*)

```
{
  boolean file_opened; /* has tfm_infile been opened? */
  font_number n; /* the number to return */
  halfword lf, tfm_lh, bc, ec, nw, nh, nd; /* subfile size parameters */
  size_t whd_size; /* words needed for heights, widths, and depths */
  int i, ii; /* font_info indices */
  int jj; /* counts bytes to be ignored */
  int z; /* used to compute the design size */
  int d; /* height, width, or depth as a fraction of design size times 2^-8 */
  int h_and_d; /* height and depth indices being unpacked */
  int tfbyte = 0; /* a byte read from the file */
  n = null_font;
  ⟨ Open tfm_infile for input 12 ⟩;
  ⟨ Read data from tfm_infile; if there is no room, say so and goto done; otherwise goto bad_tfm or
      goto done as appropriate 6 ⟩;
BAD_TFM: ⟨ Complain that the TFM file is bad 5 ⟩;
DONE:
  if (file_opened) (mp→close_file)(mp, mp→tfm_infile);
  if (n ≠ null_font) {
    mp→font_ps_name[n] = mp_xstrdup(mp, fname);
    mp→font_name[n] = mp_xstrdup(mp, fname);
  }
  return n;
}
```

5. METAPOST doesn't bother to check the entire TFM file for errors or explain precisely what is wrong if it does find a problem. Programs called TFtoPL and PLtoTF can be used to debug TFM files.

⟨ Complain that the TFM file is bad 5 ⟩ ≡

```
{
  char msg[256];
  const char *hlp[] = {"I␣wasn't␣able␣to␣read␣the␣size␣data␣for␣this␣font␣so␣this",
      "`infont'␣operation␣won't␣produce␣anything.␣If␣the␣font␣name",
      "is␣right,␣you␣might␣ask␣an␣expert␣to␣make␣a␣TFM␣file", Λ};
  if (file_opened) hlp[2] = "is␣right,␣try␣asking␣an␣expert␣to␣fix␣the␣TFM␣file";
  mp_snprintf(msg, 256, "Font␣%s␣not␣usable:␣TFM␣file␣%s", fname,
      (file_opened ? "is␣bad" : "not␣found"));
  mp_error(mp, msg, hlp, true);
}
```

This code is used in section 4.

6. ⟨ Read data from *tfm_infile*; if there is no room, say so and **goto** *done*; otherwise **goto** *bad_tfm* or **goto** *done* as appropriate 6 ⟩ ≡

```
⟨ Read the TFM size fields 7 ⟩;
⟨ Use the size fields to allocate space in font_info 8 ⟩;
⟨ Read the TFM header 9 ⟩;
⟨ Read the character data and the width, height, and depth tables and goto done 10 ⟩
```

This code is used in section 4.

7. A bad TFM file can be shorter than it claims to be. The code given here might try to read past the end of the file if this happens. Changes will be needed if it causes a system error to refer to *tfm_infile* ⊕ or call *get_tfm_infile* when *eof*(*tfm_infile*) is true. For example, the definition of *tfget* could be changed to "*begin get*(*tfm_infile*); **if** *eof* (*tfm_infile*)*then* **goto** *bad_tfm*; *end*."

```
#define tfget do
        {
          size_t wanted = 1;
          unsigned char abyte = 0;
          void *tfbyte_ptr = &abyte;
          (mp→read_binary_file)(mp, mp→tfm_infile, &tfbyte_ptr, &wanted);
          if (wanted ≡ 0) goto BAD_TFM;
          tfbyte = (int) abyte;
        }
        while (0)
#define read_two(A)
        {
          (A) = tfbyte;
          if ((A) > 127) goto BAD_TFM;
          tfget;
          (A) = (A) * °400 + tfbyte;
        }
#define tf_ignore(A)
        {
          for (jj = (A); jj ≥ 1; jj −−) tfget;
        }
```

⟨ Read the TFM size fields 7 ⟩ ≡

```
tfget;
read_two(lf);
tfget;
read_two(tfm_lh);
tfget;
read_two(bc);
tfget;
read_two(ec);
if ((bc > 1 + ec) ∨ (ec > 255)) goto BAD_TFM;
tfget;
read_two(nw);
tfget;
read_two(nh);
tfget;
read_two(nd);
whd_size = (size_t)((ec + 1 − bc) + nw + nh + nd);
if (lf < (int)(6 + (size_t) tfm_lh + whd_size)) goto BAD_TFM;
tf_ignore(10)
```

This code is used in section 6.

8. Offsets are added to *char_base*[*n*] and *width_base*[*n*] so that is not necessary to apply the *so* and *qo* macros when looking up the width of a character in the string pool. In order to ensure nonnegative *char_base* values when $bc > 0$, it may be necessary to reserve a few unused *font_info* elements.

```
⟨ Use the size fields to allocate space in font_info 8 ⟩ ≡
  if (mp→next_fmem < (size_t) bc) mp→next_fmem = (size_t) bc;      /* ensure nonnegative char_base */
  if (mp→last_fnum ≡ mp→font_max) mp_reallocate_fonts(mp, (mp→font_max + (mp→font_max/4)));
  while (mp→next_fmem + whd_size ≥ mp→font_mem_size) {
    size_t l = mp→font_mem_size + (mp→font_mem_size/4);
    font_data * font_info;
    font_info = mp_xmalloc(mp, (l + 1), sizeof (font_data));
    memset(font_info, 0, sizeof (font_data) * (l + 1));
    memcpy(font_info, mp→font_info, sizeof (font_data) * (mp→font_mem_size + 1));
    mp_xfree(mp→font_info);
    mp→font_info = font_info;
    mp→font_mem_size = l;
  }
  mp→last_fnum++;
  n = mp→last_fnum;
  mp→font_bc[n] = (eight_bits)bc;
  mp→font_ec[n] = (eight_bits)ec;
  mp→char_base[n] = (int)(mp→next_fmem − (size_t) bc);
  mp→width_base[n] = (int)(mp→next_fmem + (size_t)(ec − bc) + 1);
  mp→height_base[n] = mp→width_base[n] + nw;
  mp→depth_base[n] = mp→height_base[n] + nh;
  mp→next_fmem = mp→next_fmem + whd_size;
```

This code is used in section 6.

9. This macro is a bit odd, but it works.

```
#define integer_as_fraction(A) (int)(A)
```

```
⟨ Read the TFM header 9 ⟩ ≡
  if (tfm_lh < 2) goto BAD_TFM;
  tf_ignore(4);
  tfget;
  read_two(z);
  tfget;
  z = z * °400 + tfbyte;
  tfget;
  z = z * °400 + tfbyte;      /* now z is 16 times the design size */
  mp→font_dsize[n] = mp_take_fraction(mp, z, integer_as_fraction(267432584));
    /* times 72/72.27 2^28 to convert from TEX points */
  tf_ignore(4 * (tfm_lh − 2))
```

This code is used in section 6.

10. ⟨ Read the character data and the width, height, and depth tables and **goto** *done* 10 ⟩ ≡

```
ii = mp→width_base[n];
i = mp→char_base[n] + bc;
while (i < ii) {
  tfget;
  mp→font_info[i].qqqq.b0 = qi(tfbyte);
  tfget;
  h_and_d = tfbyte;
  mp→font_info[i].qqqq.b1 = qi(h_and_d/16);
  mp→font_info[i].qqqq.b2 = qi(h_and_d % 16);
  tfget;
  tfget;
  i++;
}
while (i < (int) mp→next_fmem) {
  ⟨ Read a four byte dimension, scale it by the design size, store it in font_info[i], and increment i 11 ⟩;
}
goto DONE
```

This code is used in section 6.

11. The raw dimension read into d should have magnitude at most 2^{24} when interpreted as an integer, and this includes a scale factor of 2^{20}. Thus we can multiply it by sixteen and think of it as a *fraction* that has been divided by sixteen. This cancels the extra scale factor contained in *font_dsize* [n.

⟨ Read a four byte dimension, scale it by the design size, store it in *font_info*[i], and increment i 11 ⟩ ≡

```
{
  tfget;
  d = tfbyte;
  if (d ≥ °200) d = d − °400;
  tfget;
  d = d ∗ °400 + tfbyte;
  tfget;
  d = d ∗ °400 + tfbyte;
  tfget;
  d = d ∗ °400 + tfbyte;
  mp→font_info[i].sc = mp_take_fraction(mp, d ∗ 16, integer_as_fraction(mp→font_dsize[n]));
  i++;
}
```

This code is used in section 10.

12. This function does no longer use the file name parser, because *fname* is a C string already.

```
⟨ Open tfm_infile for input 12 ⟩ ≡
  file_opened = false;
  mp_ptr_scan_file(mp, fname);
  if (strlen(mp→cur_area) ≡ 0) {
    mp_xfree(mp→cur_area);
    mp→cur_area = Λ;
  }
  if (strlen(mp→cur_ext) ≡ 0) {
    mp_xfree(mp→cur_ext);
    mp→cur_ext = mp_xstrdup(mp, ".tfm");
  }
  mp_pack_file_name(mp, mp→cur_name, mp→cur_area, mp→cur_ext);
  mp→tfm_infile = (mp→open_file)(mp, mp→name_of_file, "r", mp_filetype_metrics);
  if (¬mp→tfm_infile) goto BAD_TFM;
  file_opened = true
```

This code is used in section 4.

⟨ Complain that the TFM file is bad 5 ⟩ Used in section 4.
⟨ Declarations 3 ⟩ Used in section 2.
⟨ Open *tfm_infile* for input 12 ⟩ Used in section 4.
⟨ Read a four byte dimension, scale it by the design size, store it in *font_info*[*i*], and increment *i* 11 ⟩ Used in section 10.
⟨ Read data from *tfm_infile*; if there is no room, say so and **goto** *done*; otherwise **goto** *bad_tfm* or **goto** *done* as appropriate 6 ⟩ Used in section 4.
⟨ Read the TFM header 9 ⟩ Used in section 6.
⟨ Read the TFM size fields 7 ⟩ Used in section 6.
⟨ Read the character data and the width, height, and depth tables and **goto** *done* 10 ⟩ Used in section 6.
⟨ Use the size fields to allocate space in *font_info* 8 ⟩ Used in section 6.