

mf2outline

Linus Romer

March 5, 2015

Contents

1	Introduction	1
2	The <code>mf2outline.py</code> Script	2
2.1	Requirements	2
2.2	Usage and Command-line Options	2
2.3	Restrictions	4
2.4	METAFLOP	5
2.5	Other Tools	5
3	The <code>mf2outline.mp</code> Base	5
3.1	Unicode Support	5
3.2	Additional Font and Glyph Parameters	7
3.3	Ligatures and other Glyph Substitutions	10
3.4	Kerning	12

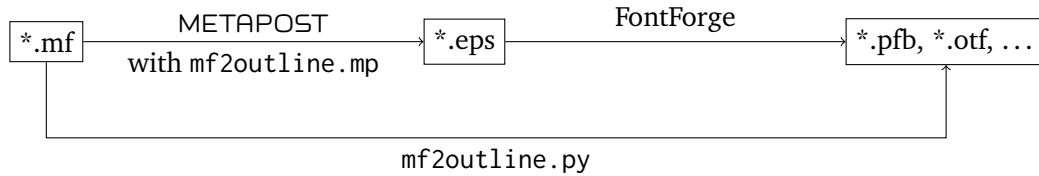
1 Introduction

METAFONT is a very versatile font description language, especially when you need to design several faces of a typeface family. However, the METAFONT compiler has some severe restrictions:

- The METAFONT compiler can only produce bitmaps and cannot produce outline font formats like Type 1 or OpenType.
- The METAFONT compiler cannot write more than 256 different characters per font.

Luckily, the METAPOST language and its compiler (see [Hobby13]) can be used as an expediant. Together with the `mfplain.mp` base, the METAPOST language supersedes nearly 100 % of the METAFONT language. The METAPOST compiler outputs PostScript files, which can be imported in FontForge and then be converted to outline font formats. This process is automated by the `mf2outline.py` script.

For compatibility reasons, the `mfplain.mp` base does not support more than 256 different characters per font. To get over this and other artificial restriction, the `mf2outline.mp` base extends the capabilities of the `mfplain.mp` base. Of course, the backwards compatibility to METAFONT will be lost by using these extensions.



2 The `mf2outline.py` Script

2.1 Requirements

The following programs have to be installed before using `mf2outline`:

- Python interpreter (`mf2outline.py` is a Python script)
- METAFONT compiler
- FontForge's python extension (`python-fontforge`)

2.2 Usage and Command-line Options

The general usage for a METAFONT file `mfsource` is easy:

```
mf2outline.py mfsource
```

This will output an OpenType font file named `mfsource.otf` in your working directory. The file extension `.mf` of the specified METAFONT source file can be omitted.

You may add some of these optional arguments:

- `-h, --help`
Show the help message and exit.
- `-v, --verbose`
Explain what is being done.
- `-vv, --veryverbose`
Explain very detailed what is being done.
- `--designsize SIZE`
Force the designsize to be `SIZE` (e.g. 12 for 12pt).
- `--raw`
Do not remove overlaps, round to int, add extrema, add hints...

--preview
 Generate only the most important letters, use icosagon pens instead of circle/elliptic pens and do not care about advanced font features like kerning and ligatures (mainly used for METAFLOP).
 List of letters: ! & () , - . / 0 1 2 3 4 5 6 7 8 9 ? A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z

-f FORMATS, --formats FORMATS
 Generate outline fonts in the formats FORMATS (comma separated list).
 Supported formats: sfd, afm, pfa, pfb, otf, ttf, eoff, svg, tfm
 Default: otf

--encoding ENC
 Force the font encoding to be ENC.
 Natively supported encodings: ot1, t1, unicode
 Default: unicode
 The file ENC.enc will be read if it exists in the same directory as the source file (the encoding name inside the encoding file must be named ENC, too).

--fullname FULL
 Set the full name to FULL (with modifiers and possible spaces).

--fontname NAME
 Set the font name to NAME (with modifiers and without spaces).

--familyname FAM
 Set the font family name to FAM.

--fullname-as-filename
 Use the fullname for the name of the output file.

--fontversion VERS
 Set the version of the font to VERS.
 Default: 001.001

--copyright COPY
 Set the copyright notice of the font to COPY.

--vendor VEND
 Set the vendor name of the font to VEND (limited to 4 characters).

--weight WGT
 Force the OS/2 weight of the font to be WGT.
 The weight number is mapped to the following PostScript weight names:
 100 Thin
 200 Extra-Light

300 Light
400 Book
500 Medium
600 Demi-Bold
700 Bold
800 Heavy
900 Black

`--width WDT`

Force the OS/2 width of the font to be WDT.

The width number stands for the following width names:

- 1 Ultra-condensed
- 2 Extra-condensed
- 3 Condensed
- 4 Semi-condensed
- 5 Medium (normal)
- 6 Semi-expanded
- 7 Expanded
- 8 Extra-expanded
- 9 Ultra-expanded

`--ffscript FFSCRIPT`

Specify an own finetuning fontforge script (e.g. `finetune.pe`). The script file has to be in the same directory as the source file. Example script:

```
Open($1);  
SelectAll();  
RemoveOverlap();  
Generate($1);  
Quit(0);
```

2.3 Restrictions

Not every valid METAFONT typeface can be automatically converted by `mf2outline`. The three most important restrictions are listed below:

- The METAFONT typeface cannot be compiled by METAPOST when it uses some special features of METAFONT that are not implemented in METAPOST (e.g. *Pandora*).
- If the font uses many overlapping filldrawn areas, FontForge does not always import the PostScript files correctly (e.g. Computer Modern). As a solution, you can use the `--raw` option and finetune the font by hand in FontForge.

- As a mathematical fact, a generic cubic beziér spline path that is drawn by a elliptic pen cannot be converted perfectly to cubic beziér spline outlines. Hence, FontForge does only an approximation job here. This approximation is normally very close to the original shape, but if you use heavily twisted cubic beziér splines, the approximation will be unsatisfactory.

2.4 METAFLOP

METAFLOP is an easy to use web application for modulating METAFONT fonts:

<http://www.metaflop.com/modulator>

The conversion to outline formats is being done by mf2outline.

2.5 Other Tools

The following two programs are alternatives to mf2outline.

mftrace is a python script that converts METAFONT fonts into Type 1 fonts. Unlike mf2outline, mftrace can cope with *every* valid METAFONT font. Unfortunately, the outline paths are not that neat.

mf2pt1 is a perl script that converts METAFONT fonts into Type 1 fonts. Actually, mf2pt1 is pretty similar to mf2outline, but does not rely that much on FontForge.

Both programs, mftrace and mf2pt1, have deeply inspired the author of mf2outline. Thus, many ideas of the two programs can be found in mf2outline, too.

3 The mf2outline.mp Base

3.1 Unicode Support

METAFONT can pack at most $2^8 = 256$ glyphs in a font. METAPOST can output nearly arbitrary many PostScript files (each containing one glyph). For compatibility reasons, METAPOST combined with mfplain.mp restricts the glyph code c to be a byte (which is a number between 0 to 255):

```
def beginchar(expr c,w_sharp,h_sharp,d_sharp) =
  begingroup
  charcode:=if known c: byte c else: 0 fi;
  charwd:=w_sharp; charht:=h_sharp; chardp:=d_sharp;
  w:=charwd*pt; h:=charht*pt; d:=chardp*pt;
  charic:=0; clearxy; clearit; clearpen; scantokens extra_beginchar;
enddef;
```

Another restriction is common to both, METAFONT and METAPOST: Numbers are represented in fixed point arithmetic as integer multiples of 2^{-16} and can (normally) not be greater than $4096 = 2^{12}$. The Basic Multilingual Plane of Unicode contains $2^{16} = 65\,536$ glyphs,

enumerated in hexadecimal numbers from 0000 upto FFFF. In `mfplain`, these hexadecimal unicode codes are represented by a string of length 4 or two byte numbers called `charcode` and `charext`. Thus, the code of the letter «J» can be represented in the following variants:

$$\underbrace{74}_{\text{decimal}} = \underbrace{"004A"}_{\text{string}} = \underbrace{\begin{pmatrix} 0 \\ 74 \end{pmatrix}}_{\text{charext/charcode}}$$

The `beginchar` macro in `mf2outline.mp` is redefined as follows:

```
newinternal string charunicode;
def beginchar(expr c,w_sharp,h_sharp,d_sharp) =
  begingroup
    charunicode:=if known c: unicode c else: "0000" fi;
    charcode:=hex(substring(2,4) of charunicode);
    charext:=hex(substring(0,2) of charunicode);
    charwd:=w_sharp; charht:=h_sharp; chardp:=d_sharp;
    w:=charwd*pt; h:=charht*pt; d:=chardp*pt;
    charic:=0; clearxy; clearit; clearpen; scantokens extra_beginchar;
  enddef;
```

There are two additional macros necessary:

- `hexadecimal` converts a decimal number to a hexadecimal string, e.g. `hexadecimal(74) = "4A"`.
- `unicode` converts a decimal number or a string to a hexadecimal string of length 4, e.g. `unicode(74) = unicode("J") = unicode("004A") = "004A"`.

The `hexadecimal` macro is defined as follows:

```
vardef hexadecimal primary n =
  save m,s;
  m:=abs round n;
  string s;
  s=
  if (m mod 16)<10:
    decimal(m mod 16)
  elseif (m mod 16)=10:
    "A"
  elseif (m mod 16)=11:
    "B"
  elseif (m mod 16)=12:
    "C"
  elseif (m mod 16)=13:
    "D"
  elseif (m mod 16)=14:
    "E"
  else:
    "F"
  fi
  ;
forever:
```

```

m:=m div 16;
exitif m=0;
s:=
if (m mod 16)<10:
  decimal(m mod 16)
elseif (m mod 16)=10:
  "A"
elseif (m mod 16)=11:
  "B"
elseif (m mod 16)=12:
  "C"
elseif (m mod 16)=13:
  "D"
elseif (m mod 16)=14:
  "E"
else:
  "F"
fi
& s;
endfor
s
enddef;

```

The unicode macro is defined as follows:

```

vardef unicode primary n =
  save s,z;
  string s,z;
  s:=
  if string n:
    if length(n)=1: % assume n to be a glyph name like "W"
      hexadecimal(ASCII n);
    else: % assume n to be a unicode like "004A" (or even "4A")
      n;
    fi
  else: % assume n to be a numeric
    hexadecimal n;
  fi
  % now fill zeroes to be a 4-digit word:
  z:=
  if length(s)<4:
    for i=1 upto (4-length(s)): "0" & endfor s;
  else:
    s;
  fi
  z
enddef;

```

3.2 Additional Font and Glyph Parameters

Unlike `mfplain.mp`, the `mf2outline.mp` base forces METAPOST to write special additional glyph information to the PostScript files and to generate an additional file `mf2outline.txt`,

that contains general font information. Normally, some of these additional information are stored in the tfm file.

There is a special version called `mf2outline-prev.mp` that causes METAPOST to use icosagon pens instead of circular/elliptic pens. The only difference to `mf2outline.mp` is the version number, that ends with the string "prev".

Example of a `mf2outline.txt` file:

```
mf2outline: font_size 10
mf2outline: font_slant 0
mf2outline: font_normal_space 2.29996
mf2outline: font_normal_shrink 0.80002
mf2outline: font_x_height 4.34998
mf2outline: font_quad 6.90002
mf2outline: font_os_weight 400
mf2outline: font_os_width 5
mf2outline: font_identifier FQDR
mf2outline: font_coding_scheme Unicode
mf2outline: font_copyright Linus Romer, 2015
mf2outline: font_name Quindesch-Regular10
mf2outline: font_fullname Quindesch Regular 10
mf2outline: font_familyname Quindesch
mf2outline: kerningclassesl
0041
0056 0057
mf2outline: kerningclassesr
0041
0056 0057
mf2outline: kerningmatrix
0 -1.0994
-1.0994 0
mf2outline: gsubtable liga
0066 0066
FB00
mf2outline: gsubtable smcp
0068
E02A
mf2outline: gsubtable c2sc
0041
E000
0048
E02A
mf2outline: eof
```

Snippet from a PostScript file containing special `mf2outline` information:

```
% mf2outline: charwd 7.28752
% mf2outline: charht 6.49994
% mf2outline: chardp 0
% mf2outline: charic 0
% mf2outline: charcode 79
% mf2outline: charext 0
% mf2outline: charunicode 004F
```

The `tfm` file stores amongst other things the following parameters:

- Global font parameters:
 - font_size
 - font_slant
 - font_normal_space
 - font_normal_stretch
 - font_normal_shrink
 - font_x_height
 - font_quad
 - font_extra_space
 - font_identifier (normally not stored)
 - font_coding_scheme (normally not stored)
- Glyph parameters:
 - charwd (character width)
 - charht (character height)
 - chardp (character depth)
 - charic (character italic correction)
 - charcode (code number of the character)
 - charext (code extension number of the character)
 - chardx (horizontal escapement of glyph positioning)
 - chardy (vertical escapement of glyph positioning)

The `mf2outline.mp` base defines some new parameters that cannot be stored in the `tfm` format:

- Global font parameters:
 - font_os_weight
 - font_os_width
 - font_version
 - font_copyright
 - font_name
 - font_fullname
 - font_familynname
- Glyph parameters:
 - charunicode (unicode string like "004A")

3.3 Ligatures and other Glyph Substitutions

The addgsub macro can be used to define many kind of glyph substitutions, e.g.:

- `addgsub("liga")("0066","0066")("FB00")` will substitute f and f by ff as ligature
- `addgsub("smcp")("0068")("E02A")` will substitute h by E02A as a small cap substitution

All the glyph substitution data is stored in a 4-dimensional matrix `gsubtables_[i][j][n][k]` with the following indices:

i number of gsubtable (e.g. 1 for "liga")

j number of current information in `gsubtables_[i]`

n switch that is either 1 (first list of chars) or 2 (second list of chars)

k index inside the list of chars

The detailed implementation of the addgsub macro is given here:

```
def addgsub(expr table)(text first)(text second) =
beginingroup
  save i,j,k;
  numeric i,j,k; % indices for gsubtables_[i][j][n][k]
  %--- store table information ---
  if table="liga": % get index i
    i=1;
  elseif table="aalt":
    i=2;
  elseif table="swsh":
    i=3;
  elseif table="hist":
    i=4;
  elseif table="locl":
    i=5;
  elseif table="rand":
    i=6;
  elseif table="nalt":
    i=7;
  elseif table="salt":
    i=8;
  elseif table="subs":
    i=9;
  elseif table="sup":
    i=10;
  elseif table="titl":
    i=11;
  elseif table="clig":
    i=12;
  elseif table="dlig":
    i=13;
  elseif table="hlig":
    i=14;
```

```

elseif table="smcp":
    i=15;
elseif table="c2sc":
    i=16;
elseif table="pcap":
    i=17;
elseif table="c2pc":
    i=18;
elseif table="unic":
    i=19;
elseif table="ital":
    i=20;
elseif table="ordn":
    i=21;
elseif table="lnum":
    i=22;
elseif table="onum":
    i=23;
elseif table="pnum":
    i=24;
elseif table="tnum":
    i=25;
elseif table="frac":
    i=26;
elseif table="afrc":
    i=27;
elseif table="dnom":
    i=28;
elseif table="numr":
    i=29;
elseif table="sinf":
    i=30;
elseif table="zero":
    i=31;
elseif table="mgrk":
    i=32;
elseif table="ss01":
    i=33;
else:
    errmessage("The first argument of addgsub() cannot be mapped to a gsub table. I will ignore this.");
fi
j:=hex(gsubtables_[i][0][0][0])+1;
gsubtables_[i][0][0][0]:=hexadecimal j; % length of gsubtables_[i]
%--- store first chars ---
k:=0; % number of first chars in gsubtables_[i][j]
for a=first:
    k=k+1;
    gsubtables_[i][j][1][k]:=unicode a;
endfor
% number of first chars is stored at 0th position
gsubtables_[i][j][1][0]:=hexadecimal k;
%--- store second chars ---
k:=0; % number of second chars in gsubtables_[i][j]

```

```

for a=second:
  k:=k+1;
  gsubtables_[i][j][2][k]:=unicode a;
endfor
% number of second chars is stored at 0th position
gsubtables_[i][j][2][0]:=hexadecimal k;
endgroup
enddef;

```

3.4 Kerning

0th index is always used for lengths...

References

[Hobby13] John D. Hobby et al. *METAPOST – A User’s Manual*. www.tug.org/docs/metapost/mpman.pdf, 2013