mf2pt1

Produce PostScript Type 1 fonts from Metafont source

Scott Pakin, `scott+mf@pakin.org`

This file documents `mf2pt1` version 2.8, dated 6 July 2024.

Table of Contents

1 Introduction 1

1.1 Requirements 1

1.2 Installation 2

2 Usage 2

2.1 Restrictions 3

2.2 Specifying font information 4

2.3 Additional command-line options 7

2.4 Custom font encodings 9

2.5 Restoring `mfplain` defaults 10

3 Future Work 10

Acknowledgments 11

1 Introduction

METAFONT is a high-level, mathematically oriented language for producing fonts. The METAFONT interpreter produces device-dependent bitmaps, which render well at the target resolution on the target device, but poorly at other resolutions or on other devices. Adobe's PostScript Type 1 font format is the de facto font standard for printers these days. It is a vector format, which implies that it scales better than bitmaps, and it delegates the device- and resolution-dependent tweaking from the font source to the target device's PostScript renderer. However, Type 1 fonts are extremely difficult to code by hand. Usually, one uses a WYSIWYG program to design a Type 1 font. METAFONT, with its font-specific programming language, is an elegant alternative. A font designer can write reusable subroutines for repeated features, such as serifs and accents. He can define a font in terms of arbitrary parameters, such as "boldness" or "italicness", making it trivial to produce entire families of fonts from a single source (hence the "meta" in the name "METAFONT"). Ideally, we would like to design a font using the METAFONT language, but produce PostScript Type 1 output instead of bitmaps.

`mf2pt1` helps bridge the gap between METAFONT and Type 1 fonts. `mf2pt1` facilitates producing PostScript Type 1 fonts from a METAFONT source file. It is *not*, as the name may imply, an automatic converter of arbitrary METAFONT fonts to Type 1 format. `mf2pt1` imposes a number of restrictions on the METAFONT input. If these restrictions are met, `mf2pt1` will produce valid Type 1 output.

1.1 Requirements

Before using `mf2pt1`, you will need to install the following programs:

Perl	`mf2pt1` is written in Perl. You will need a Perl interpreter to run it.
MetaPost	`mf2pt1` actually uses MetaPost, not METAFONT, to produce PostScript output. Specifically, you will need the `mpost` executable and the `mfplain.mp` base file.
Type 1 Utilities	Producing properly encoded Type 1 fonts is tricky. `mf2pt1` delegates the effort to the Type 1 Utilities, specifically, to the `t1asm` program within that suite.
FontForge (optional)	FontForge is a WYSIWYG, but scriptable, Type 1 font-editing program. Although FontForge is not strictly required for `mf2pt1` to produce a Type 1 font, `mf2pt1` uses FontForge when available to autohint the generated Type 1 font, thereby making it look better especially at lower resolutions.

Perl is available from the Comprehensive Perl Archive Network (`https://www.cpan.org`); MetaPost and the Type 1 utilities are available from the Comprehensive TeX Archive Network (`https://www.ctan.org`); and FontForge is available from `https://fontforge.sourceforge.net/`. In addition, MetaPost's home page is `https://tug.org/metapost.html`, and the Type 1 utilities' home page is `https://www.lcdf.org/type/`.

Besides being useful for autohinting fonts, FontForge enables a font designer to hint fonts manually for additional fine-tuning and to convert among a variety of font formats, such as PostScript, TrueType, and X Window fonts. FontForge is strongly recommended as a complement to `mf2pt1`.

1.2 Installation

To install `mf2pt1`, move the `mf2pt1` executable someplace where your operating system can find it. If you're running Microsoft Windows, you should rename the program to `mf2pt1.pl`, so that Windows knows it's a Perl script. (Alternatively, if you have `pl2bat`, use that to produce a `mf2pt1.bat` file, which you can run as simply `mf2pt1`.)

The next step is to produce a `mf2pt1.mem` file from the supplied `mf2pt1.mp`. The command to do this differs from system to system but is usually something like the following:

```
mpost -progname=mpost -ini mf2pt1 \\dump
```

Move the resulting `mf2pt1.mem` file someplace where MetaPost can find it.

The `mf2pt1` documentation (what you're reading now) is written in Texinfo and can therefore easily be converted to a variety of formats:

PDF (`mf2pt1.pdf`)

```
texi2pdf mf2pt1.texi
```

PostScript (`mf2pt1.ps`)

```
texi2dvi mf2pt1.texi
dvips mf2pt1.dvi -o mf2pt1.ps
```

HTML (`mf2pt1.html`)

```
makeinfo --html mf2pt1.texi
```

Info (`mf2pt1.info`)

```
makeinfo mf2pt1.texi
```

N.B. The `install-info` command is a convenient way to install `mf2pt1.info` on your system.

On Unix, you may also want to generate an `mf2pt1` man page. The man page is embedded within the `mf2pt1` Perl script and can be extracted with `pod2man`:

```
pod2man --center="User Commands" --date="6 July 2024" \
  --release="v2.8" mf2pt1 > mf2pt1.1
```

You can then move `mf2pt1.1` into `/usr/man/man1` or any other man page directory. Note that the `mf2pt1` man page is fairly rudimentary. It is primarily a list of the command-line options (see Section 2.2 [Font information], page 4) and a pointer to the document that you're reading now.

2 Usage

`mf2pt1` is fairly straightforward to use. To process a METAFONT source file, merely specify the filename:

```
mf2pt1 myfont.mf
```

That simple command automatically performs all of the following operations:

1. Read `myfont.mf`.

2. Use `mpost` to convert each character to a separate Encapsulated PostScript (EPS) file named `myfont.`*`num`*.
3. Process and merge the EPS files into a single "disassembled" Type 1 font called `myfont.pt1`.
4. Run `t1asm` from the Type 1 Utilities to convert the disassembled font into a true, binary Type 1 font called `myfont.pfb`.
5. Invoke `fontforge` to apply rendering hints to `myfont.pfb` and to attempt to remove overlapping paths.

The external programs launched by `mf2pt1`—`mpost`, `t1asm`, and `fontforge`—can be overridden by setting the eponymous, uppercase environment variable. For example, invoking FontForge's predecessor, PfaEdit, instead of FontForge is simply a matter of setting the '`FONTFORGE`' environment variable to '`pfaedit`' before invoking `mf2pt1`. As a corollary, you can inhibit an `mf2pt1` external program from running by setting the corresponding environment variable to the name of a nonexistent program. Arguments can be included in the environment variable's value. Hence, defining '`MPOST`' to '`mpost -recorder`', for instance, instructs `mf2pt1` to run `mpost` with the '`-recorder`' option.

2.1 Restrictions

If `mf2pt1` sounds too good to be true, it is—somewhat. `mf2pt1` is not a general-purpose METAFONT-to-Type 1 converter. Rather, it can convert only certain METAFONT constructs. This is not a showstopper for new fonts designed with `mf2pt1` in mind, but it is unlikely that `mf2pt1` will work on an arbitrary METAFONT source file.

`mf2pt1`'s biggest restriction is that each glyph must be drawn entirely from closed paths, using METAFONT's **fill** and **unfill** commands. (`mf2pt1` has limited support for **draw** and **undraw**, but their use is currently discouraged. **filldraw** and **unfilldraw** issue a warning message and invoke **draw** and **undraw**, respectively.) The Type 1 format requires that these paths be nonoverlapping. The following are some of the alternatives for removing path overlaps:

1. Install FontForge. As part of its final step in producing a Type 1 font, `mf2pt1` instructs FontForge to replace overlapping paths with nonoverlapping paths.
2. Remove overlaps using METAFONT code within the font program itself. A `.zip` file attachment to a 6 January 2005 article (`https://tug.org/pipermail/metapost/2005-January/000080.html`) by Bogusław Jackowski on the MetaPost mailing list (subject: "Re: all intersections between two paths") includes a MetaPost library which assists with that task. The library provides a **find_outlines** command which can be used to define a path as the union of two other paths. A number of MetaPost example programs are also included in the `.zip` file.
3. Design your fonts from the beginning without using overlapping paths.

A secondary restriction is that `mf2pt1` redefines a number of Plain METAFONT commands, such as **beginchar**, **fill**, and **unfill**. METAFONT font programs which redefine or bypass these (using METAFONT primitives) will not be convertible with `mf2pt1`.

A far less severe restriction is due to `mf2pt1`'s utilizing MetaPost's METAFONT interface instead of METAFONT itself. The implication is that commands not defined by MetaPost's `mfplain.mp` cannot be handled by `mf2pt1`, either. Very few fonts will have a problem with this restriction but see the MetaPost manual for more information.

2.2 Specifying font information

METAFONT fonts normally specify a set of **fontdimen**s which provide information about a particular font that cannot otherwise be inferred. These include features such as the font's x-height, quad width, interword stretchability and shrinkability, and other design characteristics that TEX makes use of. PostScript fonts utilize a largely different set of font parameters, such as the underline position and thickness, font family name, and copyright notice. `mf2pt1` provides METAFONT commands to define the PostScript font parameters in the generated Type 1 font. These parameters should appear in the METAFONT source file as follows:

if known *ps_output*:
. . .
fi

`ps_output` is defined by `mf2pt1` but not by Plain METAFONT. Checking if it is known is the recommended way to determine if the font is being built under `mf2pt1`.

The following list presents all of the font-information commands provided by `mf2pt1` and describes what each command means. Commands marked with an asterisk are also defined by Plain METAFONT and therefore do not need to be enveloped within a test for `ps_output`.

font_coding_scheme

The mapping between character numbers and PostScript names. If this is the name of a file, `mf2pt1` expects it to contain PostScript code defining a font-encoding vector. See Section 2.4 [Custom font encodings], page 9, for an example of such a file. If **font_coding_scheme** is not the name of a file, `mf2pt1` expects it to be one of the literal strings `standard` (Adobe standard encoding), `isolatin1` (ISO Latin 1 encoding), `ot1` (TEX 7-bit encoding), `t1` (TEX 8-bit encoding), or `asis` (encoding integrated with the character programs using the **glyph_name** command as described in Section 2.4 [Custom font encodings], page 9). Anything else will generate a warning message and cause `mf2pt1` to use `standard` instead.

font_comment

A textual comment that will appear within the generated font. This is often used for copyright notices.

font_family

The family that this font belongs to. For example, "Utopia Bold Italic" belongs to the `Utopia` family.

font_fixed_pitch

Whether the font is monospaced (if **true**) or or proportionally spaced (if **false**).

font_identifier (*)

The full name of the font, e.g., `Utopia Bold Italic`.

font_name The symbolic font name, used to load the font from a PostScript document. Spaces are forbidden. Generally, the font name is of the form `family-modifiers`. For example, the font name of Utopia Bold Italic would be `Utopia-BoldItalic`.

font_size (*)

The font design size. This is specified in “sharped” units within METAFONT code or as a point size on the command line.

font_slant (*)

When specified with **font_slant**, the amount of slant per point. When specified with `--italicangle`, the angle in counterclockwise degrees from the vertical (i.e., zero for an upright font, negative for a right-slanting italic font).

font_underline_position

The vertical position at which an underline should lie, specified in “sharped” units within METAFONT code or as a number of points on the command line.

font_underline_thickness

The thickness of an underline, specified in “sharped” units within METAFONT code or as a number of points on the command line.

font_unique_id

The unique ID for this font. The ID should be between 0 and 16,777,215, with the “open” range being 4,000,000-4,999,999. All IDs not in that range are allocated by contacting Adobe’s UniqueID Coordinator. (I don’t believe a fee is involved, but I don’t know for sure.) If a unique ID is not specified, `mf2pt1` will not write a unique ID to the file. Note that Adobe no longer recommends including unique IDs in fonts.

font_version

The version number of the font. This should be of the form `MMM.mmm`, where *MMM* is the major version number and *mmm* is the minor version number.

font_weight

The font weight. For example, the font weight of Utopia Bold Italic is `Bold`.

Each of the preceding font-information commands has a command-line equivalent. Their use is discouraged but they are listed here for completeness:

font_coding_scheme	`--encoding`
font_comment	`--comment`
font_family	`--family`
font_fixed_pitch	`--fixedpitch`
font_identifier	`--fullname`
font_name	`--name`
font_size	`--designsize`

font_slant	`--italicangle`
font_underline_position	`--underpos`
font_underline_thickness	`--underthick`
font_unique_id	`--uniqueid`
font_version	`--fontversion`
font_weight	`--weight`

A special case is `--fixedpitch` which does not take an argument. Rather, you should use `--fixedpitch` as the equivalent of ‘`font_fixed_pitch true`’ and `--nofixedpitch` as the equivalent of ‘`font_fixed_pitch false`’.

The next table lists the METAFONT type and default value of each of the parameters listed in the previous table.

font_coding_scheme	**string**	`"standard"`
font_comment	**string**	`"Font converted to Type 1 by mf2pt1, written by Scott Pakin."`
font_family	**string**	(The value of **font_identifier**)
font_fixed_pitch	**boolean**	**false**
font_identifier	**string**	(The input filename, minus `.mf`)
font_name	**string**	(The value of **font_family**, plus an underscore, plus the value of **font_weight**, with all spaces removed)
font_size	**numeric**	(Must be specified or `mf2pt1` will abort with an error message)
font_slant	**numeric**	0
font_underline_position	**numeric**	-1
font_underline_thickness	**numeric**	0.5
font_unique_id	**string**	(Randomly generated in the range 4000000-4999999)
font_version	**string**	`"001.000"`
font_weight	**string**	`"Medium"`

As an example, the following METAFONT code shows the usage of all of the parameters that `mf2pt1` accepts:

```
if known ps_output:
    font_coding_scheme          "ot1";
    font_comment                "Copyright (C) 2005-2024 Scott Pakin.";
    font_family                 "Kerplotz";
    font_fixed_pitch            false;
    font_identifier             "Kerplotz Light Oblique";
    font_name                   "Kerplotz-LightOblique";
    font_size                   10pt#;     % Important to include this.
    font_slant                  1/6;
    font_underline_position     -1pt#;
    font_underline_thickness    1/2pt#;
    font_unique_id              "4112233";     % Better to omit this.
    font_version                "002.005";
    font_weight                 "Light";
fi
```

In the above, the **font_fixed_pitch** call could have been omitted, as it defaults to **false**. Also, unless you've requested a unique ID from Adobe, it's generally better not to assign **font_unique_id**; let `mf2pt1` choose a random value itself.

The same parameters can also be specified on the command line as follows:

```
mf2pt1 --encoding=ot1 --comment="Copyright (C) 2005-2024 Scott Pakin."
  --family=Kerplotz --nofixedpitch --fullname="Kerplotz Light Oblique"
  --name=Kerplotz-LightOblique --designsize=10 --italicangle=-9.5
  --underpos=-100 --underthick=50 --uniqueid=4112233 --version=002.005
  --weight=Light kerplotz.mf
```

Note that a METAFONT font slant of 1/6 is equal to a PostScript italic angle of approximately -9.5. The conversion formula is $s = -\tan\theta$, in which s is the slant and θ is the italic angle. In addition, the underline position and thickness must be multiplied by 1000/**font_size** to convert from the METAFONT units that are used within the `.mf` file to the PostScript units that are used on the command line.

2.3 Additional command-line options

In addition to the command-line options for setting font parameters that were presented in Section 2.2 [Font information], page 4, `mf2pt1` supports a `--rounding` option. While `mf2pt1` normally rounds all font coordinates to the nearest integer, `--rounding` increases coordinate precision by instructing `mf2pt1` to round instead to the nearest multiple of a given fractional number. For example, '`--rounding=0.25`' rounds the coordinate $(7.4, 10.3)$ to $(7.5, 10.25)$ while it would otherwise be rounded to the less-precise $(7, 10)$.

Large glyphs can be problematic in MetaPost and therefore in `mf2pt1`. Unlike METAFONT, MetaPost does not honor '`mag`' for magnifying dimensions. Rather, the number of PostScript (a.k.a. "big") points per pixel—'`bpppix`'—is hardwired to 0.02 and all other dimensions ('`mm`', '`in`', '`pt`', etc.) are expressed in terms of that. Consequently, glyphs that multiply a large number of font units by a dimension are likely to exceed 4096, the largest value that MetaPost can represent in its fixed-point format. If numerical overflow becomes a problem you can use the `--bpppix` option to `mf2pt1` to change the value of '`bpppix`'. For example, specifying '`--bpppix=0.2`' enables a tenfold increase in maximum glyph size (with a corresponding decrease in precision).

After generating a Type 1 font, `mf2pt1` runs it through FontForge to add hinting information, remove overlaps, and otherwise clean up the generated font. The `--ffscript` option specifies the filename of a FontForge script to use instead of the default script, which is listed below:

```
Open($1);
SelectAll();
RemoveOverlap();
AddExtrema();
Simplify(0, 2);
CorrectDirection();
Simplify(0, 2);
RoundToInt();
AutoHint();
Generate($1);
Quit(0);
```

See the FontForge documentation for an explanation of each of those commands and a description of other commands made available to FontForge scripts. `mf2pt1` runs the script with one argument (`$1`), which is the name of the generated `.pfb` file.

`mf2pt1` also supports a `--help` option which summarizes all of the program's command-line options.

2.4 Custom font encodings

Section 2.2 [Font information], page 4, lists the font encodings currently supported by `mf2pt1` and mentions that an alternate encoding can be specified by providing the name of an encoding file. To elaborate, the following example represents the contents of an encoding file which defines a—largely useless nowadays—EBCDIC font encoding:

```
% Sample encoding vector: EBCDIC

% The encoding vector must have a name and be defined as a
PostScript array.
/ebcdic_encoding [
/_a0 /_a1 /_a2 /_a3 /_a4 /_a5 /_a6 /_a7 /_a8 /_a9 /_a10 /_a11
/_a12 /_a13 /_a14 /_a15 /_a16 /_a17 /_a18 /_a19 /_a20 /_a21 /_a22
/_a23 /_a24 /_a25 /_a26 /_a27 /_a28 /_a29 /_a30 /_a31 /_a32 /_a33
/_a34 /_a35 /_a36 /_a37 /_a38 /_a39 /_a40 /_a41 /_a42 /_a43 /_a44
/_a45 /_a46 /_a47 /_a48 /_a49 /_a50 /_a51 /_a52 /_a53 /_a54 /_a55
/_a56 /_a57 /_a58 /_a59 /_a60 /_a61 /_a62 /_a63 /space /_a65
/_a66 /_a67 /_a68 /_a69 /_a70 /_a71 /_a72 /_a73 /bracketleft
/period /less /parenleft /plus /exclam /ampersand /_a81 /_a82
/_a83 /_a84 /_a85 /_a86 /_a87 /_a88 /_a89 /bracketright /dollar
/asterisk /parenright /semicolon /asciicircum /hyphen /slash
/_a98 /_a99 /_a100 /_a101 /_a102 /_a103 /_a104 /_a105 /bar /comma
/percent /underscore /greater /question /_a112 /_a113 /_a114
/_a115 /_a116 /_a117 /_a118 /_a119 /_a120 /_a121 /colon /numbersign
/at /quoteright /equal /quotedbl /_a128 /a /b /c /d /e /f /g /h
/i /_a138 /_a139 /_a140 /_a141 /_a142 /_a143 /_a144 /j /k /l /m
/n /o /p /q /r /_a154 /_a155 /_a156 /_a157 /_a158 /_a159 /_a160
/asciitilde /s /t /u /v /w /x /y /z /_a170 /_a171 /_a172 /_a173
/_a174 /_a175 /_a176 /_a177 /_a178 /_a179 /_a180 /_a181 /_a182
/_a183 /_a184 /quoteleft /_a186 /_a187 /_a188 /_a189 /_a190 /_a191
/braceleft /A /B /C /D /E /F /G /H /I /_a202 /_a203 /_a204 /_a205
/_a206 /_a207 /braceright /J /K /L /M /N /O /P /Q /R /_a218 /_a219
/_a220 /_a221 /_a222 /_a223 /backslash /_a225 /S /T /U /V /W /X /Y
/Z /_a234 /_a235 /_a236 /_a237 /_a238 /_a239 /zero /one /two /three
/four /five /six /seven /eight /nine /_a250 /_a251 /_a252 /_a253
/_a254 /_a255
% Don't forget to end the array.
] def
```

All entries in the encoding vector are PostScript “names” and therefore must be prefixed with a slash. Unnamed characters such as control characters are commonly named ‘`_a`*number*’, where *number* is the decimal offset into the character table. Undefined characters are indicated by ‘`.notdef`’. In the EBCDIC example, the character at position 0 in the font will be named ‘`a0`’; the character at position 1 will be named ‘`a1`’; the character at position 74 will be named ‘`bracketleft`’; the character at position 129 will be named ‘`a`’; and so forth.

Individual characters can be mapped to an encoding either numerically or by executing a **glyph_name** command within a character definition. For example, the following code overrides the character position passed to **beginchar** (i.e., 123) with whatever character position the current encoding has designated for the '`ohungarumlaut`' glyph:

```
beginchar (123, cap_height#-1/2pt#, cap_height#, 0);
  if known ps_output:
    glyph_name "ohungarumlaut";
  fi
  . . .
endchar;
```

2.5 Restoring `mfplain` defaults

`mf2pt1` normally redefines **filldraw** as **fill** and **unfilldraw** as **unfill** because Type 1 fonts don't allow overlapping curves. Similarly, `mf2pt1` redefines **pencircle** as a 20-gon to coerce MetaPost into drawing it using a filled instead of a stroked PostScript path.

If you know you'll be postprocessing your fonts using FontForge, which can automatically convert overlapping paths to nonoverlapping paths, then you can restore the original `mfplain.mp` definitions of **filldraw**, **unfilldraw**, and **pencircle** as follows:

```
if known ps_output:
  pencircle := mfplain_pencircle;
  let filldraw := mfplain_filldraw;
  let unfilldraw := mfplain_unfilldraw;
fi
```

3 Future Work

One avenue for future work is to enable the font designer to specify Type 1 font hints directly in the METAFONT font program. Hinting is a way for a font designer to specify how a font should be rendered at low resolutions, for example, at typical monitor resolutions. In METAFONT, this is done by controlling the way that points are mapped to pixel locations, using commands such as **define_corrected_pixels**, **define_blacker_pixels**, and **lowres_fix**. Type 1 fonts are hinted in a completely different manner. Type 1 hints distinguish key character features, such as stems and dots, from decorations which can be discarded at low resolutions. The PostScript interpreter uses that information to determine how to map points to pixels. Although `mf2pt1` employs FontForge to autohint the fonts it generates, the approach is mechanical and unlikely to hint as well as a human can do manually. It would be convenient for `mf2pt1` one day to provide METAFONT commands for **hstem**, **vstem**, **dotsection**, and the other Type 1 hints. That way, hints will no longer need to be re-added manually every time `mf2pt1` regenerates a Type 1 font.

Another future addition to `mf2pt1` is the addition of more font encodings. The following are the encodings that `mf2pt1` will most likely accept:

`TeXMathItalicEncoding`

> Upper- and lowercase Greek and Latin letters, old-style digits, and a few symbols and accents.

`TeXMathSymbolEncoding`
A variety of symbols, as well as calligraphic Latin majuscules.

`TeXMathExtensionEncoding`
Variable-sized symbols, such as braces, integrals, and radicals.

`AdobeExpert`
Small caps, currency symbols, old-style digits, and various superior and inferior letters and digits.

Acknowledgments

Many thanks are owed to Werner Lemberg for writing the code underlying the `--rounding` option; for introducing the **glyph_name** command and `asis` encoding vector; for providing the default FontForge script listed in Section 2.3 [Additional command-line options], page 7; for fixing some code-generation bugs in `mf2pt1`; for finding a more robust way to dump `mf2pt1.mem`; for directing me to the overlap-removal code mentioned in Section 2.1 [Restrictions], page 3; for supplying the *crossproduct*, *makeline*, and *is_clockwise* functions used by `mf2pt1.mp`; for proposing the `--bpppix` command-line option; and for suggesting numerous features and improvements to `mf2pt1`. Thanks also go to Werner Lemberg and Bogusław Jackowski for updating the **is_clockwise** function in `mf2pt1.mp` to work around a bug in MetaPost which relates to the **turningnumber** command. Finally, thanks go to Ralf Stubner for providing the T1 encoding vector and for pointing out a typo in the generated font programs; to Michael Zedler for alerting me to problems with `--encoding` and suggesting I modify that option to accept PostScript `.enc` files; and to Michael Ummels for fixing a couple of bugs that made character widths in the generated fonts slightly too large.