

MFF.STY: Computer Modern Typefaces as the Multiple Master Fonts

Version 1.21

A.S.Berdnikov S.B.Turtia
berd@ianin.spb.su turtia@ianin.spb.su


We would like to express our warmest thanks to Dr. A.Compagner from the Delft University of Technology who spent a lot of his time and efforts trying to transform two naive students from Russia into serious scientists.

Abstract

The style file MFF.STY simulates the effect of *Multiple Master Fonts* created by Adobe using the Computer Modern typefaces as a template. It enables to vary continuously in a wide range the shape of T_EX fonts and create the unique font which suites the User's demands. Although originally MFF.STY was created for internal purposes to investigate the possibilities hidden inside the METAFONT source code for Computer Modern typefaces, it might be useful for professional applications too. The style file works correctly with L^AT_EX 2_ε as well as with L^AT_EX 2.09.

1 Introduction

The *Multiple Master Font* format for PostScript fonts was suggested some time ago by the well-known company *Adobe Inc.* It enables to vary continuously the font characteristics (say, *weight* (“boldness”) from light to black, *width* from condensed to expanded, etc.) and create the unique font which suites the particular User's demands. Like many other “new inventions” in computer assisted typography, the roots of this idea can be found inside T_EX¹ — namely, in METAFONT as the language for font description, and in Computer Modern font family created by D.Knuth in 1977–1985.

¹For example, *Microsoft Word* 6.0 was announced as the *first* program which enables to mark some place in the text by a special marker and then to refer to its position in a form: “see page ...” .

The METAFONT source code was created, and the *Computer Modern* font parameters were selected using the advises of such professional font designers as Hermann Zapf, Matthew Carter, Charles Bigelow and others (see [1] for the *full* list of contributors). The METAFONT code for *Computer Modern* typefaces has the following essential features:

- the parameter files are separated from the main source code so that the font parameters can be varied easily;
- the role of the font parameters, the details of the METAFONT source code, etc., are documented in [1] in deep details;
- the font variations (provided that the fonts and all changed .mf files have the names **different** from that for the original CM fonts) are encouraged by the author.

The *Computer Modern* fonts are parametrized using sixty two (!!!) parameters most of which are independent. It can be seen easily that such big amount of free parameters exceeds the flexibility of *any* multiple master font which is created up to now or even *will be* created by somebody in future.

The continuous parametrization of the canonical Computer Modern typefaces created by John Sauter and Karl Berry (SAUTER fonts [2]), and, on a different basis, by Jörg Knappen and Norbert Schwarz for *European Computer Modern* typefaces [3], enables to vary continuously the font size in a wide range without loosing the high quality of the output. As a result it is easy to manipulate with *Computer Modern* typefaces like the Adobe multiple master fonts, and to produce an enormous amount of **freeware** fonts of professional quality.

The style file MFF.STY described here performs this work. It follows the ideas implimented in the package MFP_C² and enables to specify the new fonts inside L^AT_EX document without dealing with the details of METAFONT programming and manual specification of each of 62 parameters used in *Computer Modern* source files. The User can variate the font shape continuously between CMR, CMBX, CMSL, CMSS, CMTT and CMFF font families, specify the *weight*, *width*, *height* and *contrast* of the output font independently, and in addition he/she can play the character characteristics so that the output does not look like the canonical *Computer Modern* typefaces at all.

2 Main Command: define a font

The generation of the .mf header file and the activation of the new font is performed by the command `\MFFgener` which has the format:

$$\text{\MFFgener [fntscaled] {\fntcmd}\{filename}\{fntsize}}$$

²Like MFP_C, the first pass of L^AT_EX creates the .mf file, then the .mf file is processed by METAFONT, and at the second pass of L^AT_EX the new font is used to make the output.

The command `\fntcmd` will switch inside the document to the desired font like it is done by the \LaTeX commands `\bf`, `\sl`, `\sf`, etc.³ The file `filename.mf` will contain the METAFONT source code of the new font⁴. The value `fntsize` specifies the design size of the new font (it defines the size of a new font and is used by MFF.STY to calculate the font parameters as the functions of this reference value — see sections 3 and 4 for more details). The optional parameter `fntscaled` specifies the additional scaling of this font in \LaTeX document. Examples:

```
\MFFgener{\zfnt}{z mz30}{30pt}
\MFFgener[scaled 2000]{\zfnt}{z mz15}{15pt}
\MFFgener[at 30pt]{\zfnt}{z mz20}{20pt}
```

The `.mf` file created by MFF.STY contains only the list of 62 font parameters which determine the character shapes. It also contains the command that loads the driver file (`roman.mf`, `csc.mf`, etc.), and the driver file assembles all the necessary METAFONT source code. It is assumed that the driver files for the fonts described in section 6 together with all necessary `.mf` files are already installed on your computer.

When the file `filename.mf` is created, it is necessary to process it by METAFONT to get the metric file `filename.tfm` and the bitmap file `filename.pk`. The second pass of \LaTeX will use the metric information to format the document properly, and the *DVI*-drivers will use the bitmap data to make the output. The proper configuration of \TeX -compiler and *DVI*-drivers so that they can find these files is the User's task (please consult your local \TeX -expert how to configure it⁵).

If \LaTeX can find the file `filename.tfm`, it assigns the font `filename` to the command `\fntcmd` which can be used later in the document to switch to the desired font. If \LaTeX cannot find the `.tfm` file, the warning message

```
No file filename.tfm -- dummy font will be used
```

is displayed which means that the text typed by this new font will disappear from the *DVI*-file (dummy font is the artificial font which contains no characters).

The file `filename.mf` which contains the METAFONT data necessary to make the font is created after *each* pass of \LaTeX and the warning message

```
Do not forget to process filename.mf and reprocess this file
```

is displayed even if the new `.mf` file is identical to the previous one. It is the User's responsibility to guarantee that the MFF.STY commands are the same as during the previous pass of \LaTeX and that the `.tfm` and `.pk` files are produced from the correct version of `.mf` file.

³The difference is that the size of this font is not influenced by the commands like `\large` or `\small` — it is fixed and is determined totally by the parameters `fntscaled` and `fntsize`.

⁴To minimize the attempts to create the fonts with the names already used in Computer Modern family, etc., the prefix `xx` is added before the file name automatically. See section 11 for more details.

⁵Some advises how to configure the programs if you use *MS DOS/emTeX* are in section 9.

3 Mixture of independent fonts

The *Computer Modern* font sequences

roman, **bold**, *slanted*, sans serif, typewriter, funny,
dunhill, quotation

share just the same METAFONT source code but with different values for font parameters. Since each font exists at different sizes, it is possible to construct for each sequence and for each parameter a continuous approximation which is the function of the font size. Such approximations, for example, enable to generate fonts with the font sizes different from that created by D.E.Knuth.

At least two *ready-to-use* font approximations are available from CTAN. One is the SAUTER font package created by John Sauter and Karl Berry [2], and the other is realized in DC and TC fonts created by Jörg Knappen and Norbert Schwarz [3]. Special MFF.STY comands control the approximation scheme:

`\MFFsauter` — SAUTER-type approximation is used;
`\MFFdcfonts` — DC FONT approximation is used;

where the default mode is `\MFFsauter`.

The continuous approximation for CMR parameters enables to create the METAFONT header file for CMR font with an arbitrary font size. Similar approximations and, as a result, the font headers with an arbitrary font size, can be constructed for CMBX, CMTT, CMSS, etc. The main hypothesis used below is the following: if some font headers result to correct fonts when processed by METAFONT, the font header constructed from the *weighted sum* of the font parameters extracted from these font headers gives the correct font also⁶. Although it is not necessarily true, the experiments with *Computer Modern* typefaces show that if the header files corresponding to different *Computer Modern* fonts *with the same font size* are selected, this assumption is fulfilled with great probability.

The fonts CMR, CMBX, CMTT, CMSS, etc., have quite different character styles although they use just the same METAFONT driver file `roman.mf`. The CMTT fonts have nearly rectangular serifs, nearly no contrast between thin and thick lines, and different ratio width/height as compared with CMR. The CMSS fonts has no serifs at all, they also have no contrast, the thickness of their lines is greater than that for CMTT. Other fonts have their own specific features, but inspite of this fact they can be “added” together — at least mathematically as the 62-dimensional vectors of font parameters. The resulting font is no longer CMR, CMBX, CMTT, etc., but something intermediate with a unique shape.

Taking into account the New Font Selection Scheme used by $\text{\LaTeX} 2_{\epsilon}$, it is preferable to decompose the CMBX font sequence into *two* sequences — one

⁶The internal relations between font parameters described in [1] are conserved for the weighted sum of font parameters.

for “boldness” (*weight*) and one for “extension” (*width*) characteristics. The `MFF.STY` uses the special fonts `CMB'` (bold as `CMBX` and wide as `CMR`) and `CMX` (wide as `CMBX` and bold as `CMR`) which are derived from `CMBX`.

The mixture of fonts is performed by the command

```
\MFFmixture{\alpha_1}{\alpha_2}{\alpha_3}{\alpha_4}{\alpha_5}{\alpha_6}
```

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6$ specify the weight factors: α_1 corresponds to `CMB`, α_2 corresponds to `CMX`, α_3 corresponds to `CMSS` (sans serif fonts), α_4 corresponds to `CMTT` (typewriter fonts), α_5 corresponds to `CMFIB` (“Fibonacci” fonts), α_6 corresponds to `CMFF` (funny fonts)⁸.

If some parameter has the value p_{cmr} for `CMR` font, the value p_{cmb} for `CMB` font (with the same font size!), the value p_{sf} for `CMSS` font, etc., the *mixture* value p_* for this parameter is calculated as

$$p_* = \alpha_0 p_{cmr} + \alpha_1 (p_{cmb} - p_{cmr}) + \alpha_2 (p_{cmx} - p_{cmr}) + \alpha_3 (p_{sf} - p_{cmr}) + \alpha_4 (p_{tt} - p_{cmr}) + \alpha_5 (p_{fib} - p_{cmr}) + \alpha_6 (p_{funny} - p_{cmr}) \quad (1)$$

The value α_0 is set to 1.0 by the command `\MFFmixture`, but the User can assign an arbitrary value to it using the commands `\setMFF[\alpha_0]{cmr}` and `\mixMFF[\alpha_0]{cmr}` which are described below.

This procedure enables, for example, to make the font “less bold” than `CMR` or “more bold” and “more extended” than `CMBX` using the weight factors which are less than 0 or greater than 1, and to create the “mutant” combinations of nearly incompatible font families⁹.

The command `\MFFmixture{\alpha_1}{\alpha_2}{\alpha_3}{\alpha_4}{\alpha_5}{\alpha_6}` is equivalent to the following sequence of commands:

```
\clearMFF      % synonym for \MFFmixture{0}{0}{0}{0}{0}{0}
\mixMFF[\alpha_1]{bold}
\mixMFF[\alpha_2]{x}
\mixMFF[\alpha_3]{sf}
\mixMFF[\alpha_4]{tt}
\mixMFF[\alpha_5]{fib}
\mixMFF[\alpha_6]{funny}
```

These commands are better recognized due to their mnemonic form, they can be specified in an arbitrary order, the commands with zero α 's can be skipped since the zero weight factors are already assigned by `\clearMFF`. If several commands with the same font name are encountered, the last specification is active. The default value for the optional parameter α is 1.0.

⁷It is not the same as `cmb10` created by D.E.Knuth — the font parameters are different.

⁸The fonts *slanted*, *quotation* and *dunhill* are not included in this list because they can be produced from `CMR` easily using the scaling commands described in section 4.

⁹Caution: playing this game it is very easy to get the `.mf` file which cannot be passed through `METAFONT` without errors if you do not understand clearly what are you doing.

The mixture between CMR and a *single* font can be specified by a single mnemonic command selected from

```
\setMFF[\alpha_1]{bold}   \setMFF[\alpha_3]{sf}   \setMFF[\alpha_5]{fib}
\setMFF[\alpha_2]{x}     \setMFF[\alpha_4]{tt}   \setMFF[\alpha_6]{funny}
```

which is equivalent to `\MFFmixture` where only one weight factor has non-zero value. If the optional parameter α is skipped, the default value 1.0 is used. It means that the “pure” font family can be specified by the commands

```
\setMFF{bold}   \setMFF{sf}   \setMFF{fib}
\setMFF{x}     \setMFF{tt}   \setMFF{funny}
```

Although L^AT_EX 2_ε/NFSS considers the *weight* and the *width* as two independent font characteristics, the movement in the direction “bold + extended” (i.e., to CMBX font family) usually gives more pleasant results. To perform this operation it is necessary to assign the equal weights to α_{cmb} and α_{cmx} which can be done by the mnemonic commands

```
\mixMFF[\alpha_{bx}]{bx}   and   \setMFF[\alpha_{bx}]{bx}
```

The commands

```
\mixMFF[\alpha_{cmr}]{cmr}   and   \setMFF[\alpha_{cmr}]{cmr}
```

play a special role: they enable to assign the value $\alpha_0 = \alpha_{cmr}$ in the equation (1) (usually α_0 is set to 1.0 by the command `\MFFmixture` — for the evident reasons).

4 Modifications of font parameters

You can use the weighted mixture of font ingredients using the commands `\setMFF`, `\mixMFF` and `\MFFmixture` described above, but you can also vary “by hand” the font parameters which control the essential details of the character shape.

4.1 Variations of the height

The following commands enable to vary the height of the vertical elements of the characters:

- `\MFFscaleHeight{factor}` — scale proportionally the height and the depth of the characters;
- `\MFFscaleAsc{factor}` — scale the height of the capital characters, brackets, digits, etc., and the ascenders of the characters like ‘b’, ‘h’;

`\MFFscaleDesc{factor}` — scale the depth of comma and the descenders of the characters like ‘Q’, ‘y’;

`\MFFscaleMath{factor}` — scale the height of digits and the height of the horizontal bar (the middle line) for mathematical symbols like =, +, −.

If several height factors are specified, their effect is combined. The curious font `cmdunh10` can be reproduced *exactly* by proper specification of all these factors.

Example:

```
\MFFscaleAsc{1.5}
\MFFscaleDesc{1.2}
```

4.2 Variations of the width

The *width* of CMR font can be varied by mixturing with the $\overline{\text{CMX}}$ font. It increases the character width and also performs some *fine* tuning of other font parameters. Due to this reason the mixturing with $\overline{\text{CMX}}$ can be advantageous to vary the width of characters if you deal with CMR family. From the other side, for CMTT fonts or CMSS fonts the mixturing with $\overline{\text{CMX}}$ results also to some variation of the character shapes which can be an undesirable effect.

The User can specify the explicit width multiplication which means that the font parameters which define the width of the characters are multiplied by some factor:

```
\MFFscaleWidth{factor}
```

After this command the font parameters

```
u#, serif_fit#, cap_serif_fit#, jut#, cap_jut#
```

are scaled proportionally to *factor*. The values *factor* > 1.0 correspond to expansion, the values 0 < *factor* < 1.0 — to compression of the characters. The command `\MFFscaleWidth{1.0}` restores the default width.

4.3 Variations of the weight

The *weight*, i.e., the “boldness” of the characters can be controlled by mixturing with $\overline{\text{CMB}}$. Similarly to *width* correction described in the previous section, for such fonts as CMTT or CMSS it is accompanied by some undesirable changes in the character shapes.

The following commands control explicitly the *weight* and the *contrast* of the characters:

`\MFFscaleBoldLines{coef1}` — scale the thickness of *thick* strokes by *coef₁*;

`\MFFscaleThinLines{coef2}` — scale the thickness of *thin* strokes by *coef₂*.

The coefficient $coef_1$ scales the values of the font parameters

`stem#`, `curve#`, `ess#`, `flare#`, `dot_size#`, `cap_stem#`, `cap_curve#`,
`cap_ess#`, `bar#`, `slab#`, `cap_bar#`, `cap_band#`, `thin_join#`.

The coefficient $coef_2$ scales the values of font parameters

`hair#`, `vair#`, `cap_hair#`, `rule_thickness#`, `notch_cut#`,
`cap_notch_cut#`.

Multiplication by $coef_1$ and $coef_2$ increases the font contrast in $coef_1/coef_2$ times. The contrast can be specified explicitly using the commands

`\MFFcontrast[type]` or `\MFFcontrast{value}`

which defines the parameter *value* equal to the ratio of the thickness of *thin strokes* to the thickness of *thick strokes*. The commands work as:

`\MFFcontrast[s]` — no correction of the contrast;
`\MFFcontrast[n]` — no contrast at all ($value=1$);
`\MFFcontrast[d]` — 50% contrast ($value=0.5$);
`\MFFcontrast{value}` — the *value* is specified explicitly.

If the correction of the contrast is active (no command `\MFFcontrast[s]`), all thick element of lowercase characters are equal to `stem#`, all thick element of uppercase characters are equal to `cap_stem#`, all thin elements have the thickness of `stem#` or `cap_stem#` multiplied by *value*, the values `stem#` and `cap_stem#` are multiplied by $coef_1$, the value $coef_2$ is ignored.

4.4 Miscellaneous variations

The following scaling factors can help to perform fine tuning of the characters (see [1] for more details):

`\MFFscaleJoinLines{factor}` — variable `thin_join#` is multiplied by *factor* (this variable is responsible for fine connection between thin and thick lines in ‘h’, ‘m’, ‘n’);

`\MFFscaleNotchCut{factor}` — variables `notch_cut#` and `cap_notch_cut#` are multiplied by *factor* (these variables are responsible for sharp corners in letters ‘A’, ‘V’, ‘w’);

`\MFFscaleDotSize{factor}` — variables `dot_size#` and `flare#` are multiplied by *factor* (these variables are responsible for dots in ‘i’, ‘.’ and bulbs in ‘a’, ‘c’);

`\MFFscaleSerifDish{factor}` — variable `dish#` is multiplied by *factor* (this variable defines the curved shape of the serif platform).

Since these parameters perform fine tuning of the character shape, it can be desirable to assign the specific value to some of them instead of scaling the default value. This operation requires the “expert level macros” described in section 4.7. For example, to assign a very big number to the variables `notch_cut#` and `cap_notch_cut#` which control the sharpness of the corners in letters ‘A’, ‘V’, ‘w’, etc.¹⁰, the following commands can be used:

```
\MFFcatcode
\def\MFF@assign@notch_cut{\@tempdimb=16383pt}
\def\MFF@assign@cap_notch_cut{\@tempdimb=16383pt}
\noMFFcatcode
```

To return the rule of calculation for these parameters to the default state, the commands

```
\MFFcatcode
\def\MFF@assign@notch_cut{}
\def\MFF@assign@cap_notch_cut{}
\noMFFcatcode
```

should be used.

4.5 Slanted characters

The inclination of the characters is defined by the variable `slant#` which is specified explicitly by the commands:

```
\MFFslant{parm} — set slant as a fraction;
\MFFslantD{parm} — set slant as an angle specified in degrees.
```

For example, *slant* typical for CMSL is specified as `\MFFslant{1/6}`, and *slant* typical for CMSSI is specified as `\MFFslantD{12}` which is just the same as `\MFFslant{ $\text{sind}(12)/\text{cosd}(12)}$ }`. The arguments of the commands `\MFFslant` and `\MFFslantD` are interpreted as the text strings directly transferred to `.mf` files, not as the numerical values.

4.6 Logical flags

The logical switches usually specified at the end of `*.mf` file can be controlled by the following commands (`char=n` means *false*, `char=y` means *true*):

```
\MFFflagSquareDots{char} — set logical variable square_dots (should dots be square?);
```

¹⁰This operation is necessary to produce high-quality outlined characters using *font tricks* commands described in section 7 (see [7] for more details).

`\MFFflagHefty{char}` — set logical variable `hefty` (should we try hard not to be overweight?);

`\MFFflagSerifs{char}` — set logical variable `serifs` (should bulbs and serifs be attached?);

`\MFFflagMonospace{char}` — set logical variable `monospace` (are all characters of the same width?);

`\MFFflagVariantG{char}` — set logical variable `variant_g` (should an italic-style `g` be used?);

`\MFFflagLowAsterisk{char}` — set logical variable `low_asterisk` (should the asterisk be centered at the axis?);

`\MFFflagMathSpacing{char}` — set logical variable `math_fitting` (should math-mode spacing be used?).

The level of ligature and kerning data is specified by the command:

`\MFFflagLigs{type}` where `type` is 0, 1, 2, `s` or `n` — set ligature level: 0, 1, 2 correspond to the value of the variable `ligs`, `n` is equivalent to 0, `s` set ligature level like in CMR fonts (it is 1 if font design size is less than 6pt, and 2 otherwise).

4.7 Expert level macros

It is possible to change manually each font variable if the default value calculated by `MFF.STY` is not satisfactory. To perform such operation it is necessary to specify the macro with the name

`\MFF@assign@varname`

which redefine the value of the register `\@tempdimb`. On input the register `\@tempdimb` is equal to automatically calculated value for that parameter, and the register `\@tempdima` is equal to the font design size. On output the register `\@tempdimb` should contain the new value for the font parameter. When the macro `\MFF@assign@varname` is executed, the correction procedure described in section 5 is performed.

For example, the following macro

`\def\MFF@assign@x_height{\@tempdimb=3\@tempdimb}`

scales by 3 the font variable `x_height#`, and the macro

`\def\MFF@assign@crisp{\@tempdimb=0pt}`

assigns zero value to the font variable `crisp#`. To delete this user-specified transformation it is necessary to use the explicit dummy definition

```

\def\MFF@assign@x_height{}
\def\MFF@assign@crisp{}

```

The characters ‘_’ and ‘@’ should have the status *letters* to type such macro definition. The command `\MFFcatcode` assigns the status *letters* to ‘_’ and ‘@’, and the command `\noMFFcatcode` returns the previous status for these characters. These commands work correctly even inside `.sty` files where the character ‘@’ have the status *letter* before L^AT_EX starts to process the style file, and should conserve this status when L^AT_EX finishes to process the style file. Since the command `\noMFFcatcode` returns *the previous* status to ‘_’ and ‘@’ (that is, the catcode which they have during the last command `\MFFcatcode`), two subsequent commands `\MFFcatcode` without intermediate `\noMFFcatcode` produce an error: the status of ‘_’ and ‘@’ will be *letters* even after the command `\noMFFcatcode`. To assign the status *other characters* to ‘_’ and ‘@’ unconditionally, the command `\otherMFFcatcode` can be used.

The only operation that should be performed inside `\MFF@assign@varname` is the re-definition of `\@tempdimb`. No `\@tempdima` nor other internal variables can be changed although the contents of `\@tempdima` and the initial value of `\@tempdimb` can be used to calculate the output value. The exception are the dimensional registers `\MFF@dimenA`, `\MFF@dimenB`, `\MFF@dimenC`, `\MFF@dimenD` which can be used for intermediate calculations. Also, it is necessary to take into account that:

- the value of all dimensional `.mf` variables except

```
notch_cut#, cap_notch_cut#, rule_thickness#
```

is divided by 36 when printed to the header file;

- the values of the variables

```
notch_cut#, cap_notch_cut#, rule_thickness#
```

are divided by 100 when printed to the header file;

- the values of non-dimensional variables

```
fudge, math_spread, superness, superpull, beak_darkness
```

are printed to the header file ‘as it is’ but it is to be specified in `pt` — say, to assign the value 0.5 to the variable `fudge`, the command

```
\def\MFF@assign@fudge{\@tempdimb=0.5pt}
```

should be used;

- the value of the font parameter `slant` is set directly by the commands `\MFFslant` and `\MFFslantD` and cannot be specified using the command `\MFF@assign@slant`;
- the value of `font_size` is defined by the parameter `fntsize` of the command `\MFFgener` (see section 2) and cannot be changed after it.

5 Automatic control of font parameters

The METAFONT programs which describe the *Computer Modern* typefaces assume that the following mutual relations between font parameters are fulfilled (see [1] for more details):

- $0.5 \cdot x_height \leq bar_height \leq 0.55 \cdot x_height$
- $asc_height \geq 1.2 \cdot x_height$
 $curve \geq stem$
 $cap_stem \geq stem$
 $cap_curve \geq curve$
- each of the variables
 $thin_join, hair, vair, stem, curve, ess, flare, dot_size, bar,$
 $slab, cap_hair, cap_stem, cap_curve, cap_ess, cap_bar, cap_band$
are no less than `crisp`, `tiny` and `fine`;
- the variables `stem_corr` and `vair_corr` are no greater than $\frac{1}{5} cap_hair$, $\frac{1}{6} stem$, $\frac{1}{4} fudge \times stem$ and $\frac{1}{12} curve$;
- the variable `vair_corr` is no greater than $\frac{1}{4} slab$;
- the variable `stem_corr` is no greater than $\frac{1}{16} fudge \times hair$.

Although even the canonical header files sometimes violate these conditions, it is more safe if the font parameters calculated by `MFF.STY` satisfy these relations (especially if the *Computer Modern* driver files `roman.mf`, `textit.mf`, `csc.mf` are used). From the other side, several interesting effects can be achieved only when these relations are violated (provided that the `.mf` file is still processed by METAFONT without errors). The automatical correction is switched *on* and *off* by the commands:

- `\MFFcheck` — the conditions described above are checked and the variable values are corrected if necessary;
- `\MFFnocheck` — the automatical checking and correction of the font parameters is switched off although the condition that some critical parameters are not negative, is still checked and corrected, if necessary.

6 Font classes

The NFSS/L^AT_EX₂_ε classifies T_EX font families in a way which is different from the logical structure of .mf files for *Computer Modern* typefaces. That is, the *italic* and SMALL CAPS are at the same family `roman` together with **bold** and *slanted* fonts, although they are produced by different driver files. Similarly, `roman`, `typewriter`, `sans serif`, `dunhill`, `quotation` and `funny` fonts are considered as different font families although they are produced by the same driver file `roman.mf` (but with different font parameters).

As soon as we deal with MFF.STY there is no sharp boundary between `roman`, **bold**, *slanted*, `typewriter`, `sans serif`, `funny` and `dunhill` fonts — each font is smoothly converted to another one, while *italic* and SMALL CAPS fonts are quite different — they use different driver files. The macros MFF.STY assign different *classes* to these fonts to distinguish such difference from *font families* used in NFSS. The following font classes can be specified now:

CMR — Computer Modern Roman;

CMTI — Computer Modern Text Italic;

CMCSC — Computer Modern Small Caps;

DCR — European Computer Modern Roman;

DCTI — European Computer Modern Text Italic;

DCCSC — European Computer Modern Small Caps;

CMRZ — CMZ Computer Modern Roman/Cyrillic created by Nana Glonti and Alexander Samarin;

CMRIZ — CMZ Computer Modern Text Italic/Cyrillic;

CMCCSC — CMZ Computer Modern Small Caps/Cyrillic;

LHR — LH Computer Modern Roman/Cyrillic created by Olga Lapko and Alexey Khodulev;

LHTI — LH Computer Modern Text Italic/Cyrillic;

LHCSC — LH Computer Modern Small Caps/Cyrillic.

LLR — LL Computer Modern Roman/Cyrillic created by Olga Lapko and Alexey Khodulev (cyrillic part only);

LLTI — LL Computer Modern Text Italic/Cyrillic;

LLCSC — LL Computer Modern Small Caps/Cyrillic.

The font class is specified by the command

```
\MFFclass{class}
```

For example the command `\MFFclass{CMR}` activates *Computer Modern Roman* fonts (that is, the font header file will use the driver file `roman.mf`).

The set of font classes can be extended easily as soon as there is a font based on the same set of parameters as *Computer Modern* fonts. The only thing to do is to specify the macro which writes the `font_identifier` value and the operator `generate` with corresponding name of the driver file (see `MFF.STY` for the examples).

7 Special Effects

To make more fun some special effects described in [6, 7] can be included in your font. If you specify some *font trick* declaration, the special portion of METAFONT code is inserted in the header file which modifies the characters generated by original METAFONT subroutines. The *font trick* declarations can specify

- pattern for the main body of the character;
- pattern for the rectangular box (background) of the character;
- pattern for the character shadow (if present);
- pattern for the underlining of the character (if present) [this feature will be realized in future versions];
- additional transformations (reflections, rotations, etc.) of the characters.

The command `\MFFtrick` which specifies these attributes has the format:

```
\MFFtrick{char-style}{box-style}{shadow-style}{underline}{transform}
```

where the parameter *underline* is reserved for future improvements and means nothing in the current version. The following letters can be used to specify the *char-style*, *box-style* and *shadow-style*:

- z** — no such element or solid white pattern;
- b** — solid black pattern;
- h** — horizontal stripes;
- v** — vertical stripes;
- r** — slanted stripes /;
- l** — slanted stripes \;
- g** — rectangular grid ('h'+‘v’);
- s** — slanted grid ('r'+‘l’);
- d** — dotted grid.

The capital letters Z, B, H, V, R, L, G, S, D mean that the outline of the element contour is added to the filling pattern. The specifications **b** and **B** are equivalent since the outlined contour is undistinguishable over the solid black pattern. For example, the outlined main character with a white body, white box background and outlined shadow filled with dotted grid is specified by the command `\MFFtrick{Z}{z}{D}{}`.

The parameter *transform* specifies the transformation of the character¹¹:

rr — rotation -90° ;
rl — rotation $+90^\circ$;
ro — rotation 180° ;
sx — symmetry $x \rightarrow -x$;
sy — symmetry $y \rightarrow -y$;
sz — symmetry $(x, y) \rightarrow (y, x)$,
st — symmetry $(x, y) \rightarrow (-y, -x)$.

The font trick effects specified by the command `\MFFtrick` are used immediately by the subsequent commands `\MFFgener`. The following commands switch *on* and *of* the font trick effects provided that the font trick parameters are already established:

`\MFFfonttricks` — activates the font trick declarations;

`\MFFnotricks` — deactivates the font trick declarations.

The following commands can be used to specify the individual font trick elements (these declarations are used by the subsequent commands `\MFFgener` only after the explicit command `\MFFfonttricks` or if they are specified after the command `\MFFtrick`):

`\trickMFFchar{char-style}` — the pattern style for filling of the body of the character (“z” means white body);

`\trickMFFbox{box-style}` — the pattern style for filling of the background box (“z” means empty background);

`\trickMFFshadow{shadow-style}` — the pattern style for filling of the character shadow (“z” means no shadow);

`\trickMFFtransform{transform}` — the additional transformation of the character.

Some typical font trick effects can be specified using the command

¹¹Several transformations can be used — in this case the individual letters are separated by commas like in `{rr,sx,ro}`.

`\MFFstdtrick{trick-name}`

where the following *trick-name* identifiers can be used:

- `standard` — no font tricks (`=\MFFtrick{b}{z}{z}{}`);
- `reversed` — *reversed* characters: white letters over black rectangle;
- `dotted` — characters with a body filled with dots;
- `striped` — characters with a body filled with rectangular grid;
- `stripedH` — characters with a body filled with horizontal stripes;
- `stripedV` — characters with a body filled with vertical stripes;
- `slanted` — characters with a body filled with slanted rectangular grid;
- `slantedL` — characters with a body filled with slanted stripes “\”;
- `slantedR` — characters with a body filled with slanted stripes “/”;
- `outlined` — outlined white characters;
- `shadowed` — outlined white characters with solid shadows;
- `sShadowed` — outlined white characters with outlined white shadows;
- `shadowonly` — only the solid shadow of the character is still present.

The parameters of the *font trick* effects are controlled by the commands:

- outline contour parameters:
 - `\stepMFFoutline{value}` — the thickness of the outline contour.
- shadow parameters:
 - `\stepMFFshadow{value}` — the step of the shadow shift;
 - `\cornerMFFshadow{char}` — the corner of the shadow:
 - A — right/down corner;
 - B — right/upper corner;
 - C — left/upper corner;
 - D — left/down corner.
- the parameters for the patterns which are used for the main body of the character:
 - `\stepMFFcharpattern{value}` — the step between lines and dots;
 - `\penMFFcharpattern{value}` — the thickness of lines and dots.

- the parameters for the patterns which are used for the background of the character:

`\stepMFFboxpattern{value}` — the step between lines and dots;
`\penMFFboxpattern{value}` — the thickness of lines and dots.

- the parameters for the patterns which are used for the character shadow:

`\stepMFFshadowpattern{value}` — the step between lines and dots;
`\penMFFshadowpattern{value}` — the thickness of lines and dots.

The values of the thickness/step size for the striped, slanted and dotted patterns are specified using the non-dimensional *value* which is the factor applied to the (dimensional) font parameter `hair#`. For example, the command `\stepMFFboxpattern{0.5}` specifies that the step size between horizontal, vertical or slanted lines used to construct the pattern for the background is equal to $0.5 \times \text{hair#}$.

8 Default state

The default values for *all* parameters used by MFF.STY are assigned by the command `\MFFdefault` which is defined as

```
\def\MFFdefault{
%
  \MFFcontrast[s]
  \MFFscaleBoldLines{1} \MFFscaleThinLines{1}
  \MFFscaleWidth{1} \MFFscaleHeight{1}
  \MFFscaleAscend{1} \MFFscaleDescend{1} \MFFscaleMath{1}
  \MFFscaleJoinLines{1} \MFFscaleNotchCut{1}
  \MFFscaleDotSize{1} \MFFscaleSerifDish{1}
%
  \MFFslant{0}
  \MFFflagLigs{s} \MFFflagMonospace{n}
  \MFFflagSquareDots{n} \MFFflagHefty{n}
  \MFFflagSerifs{y} \MFFflagVariantG{n}
  \MFFflagLowAsterisk{n} \MFFflagMathSpacing{n}
%
  \stepMFFoutline{0.075}
  \stepMFFshadow{0.5}
  \cornerMFFshadow{A}
  \stepMFFbackpattern{0.375} \penMFFbackpattern{0.075}
  \stepMFFcharpattern{0.375} \penMFFcharpattern{0.075}
  \stepMFFshadowpattern{0.375} \penMFFshadowpattern{0.075}
%
  \MFFstdtrick{standard}
```

```

        \MFFnotricks
%
        \MFFsauter
        \MFFclass{CMR}
        \MFFmixture{0}{0}{0}{0}{0}{0}
        \MFFnocheck
}

```

It can be called at any moment to initialize from the very beginning the MFF.STY parameters.

9 Configuration of emTeX programs

The way you should setup TeX and METAFONT to work with MFF.STY correctly depends on your local system. The most difficult (and system-dependent) aspect is how to teach METAFONT to generate the *.pk-files according to your printer specification, and how to teach TeX and DVI-drivers to find METAFONT's output files. For MS DOS and OS/2 and for emTeX package this problem can be solved using the utility *MFJob* as it is described below.

The configuration used here assumes that the .mf files and .tfm files are placed at the working directory, and the .pk files are placed in its subdirectories with the names corresponding to font resolution. The .mf files created by MFF.STY are placed at the current directory automatically. The .tfm and .pk files are placed at the proper directories by the utility *MFJob* if it uses the following script file:

```

%
% file mff.mfj / script for MFPiC and MFF.STY
%
input [modes];
def s=[s0];
{
base=plain;
fonts=f; mags=s; m;
output=pk[.\@Rrdpi\@f] tfm[@f] log[@f];
}

```

The script file named *mff.mfj* should be placed at the directory `\EMTEX\MFJOB\`. To process the .mf file by METAFONT and to generate the output font for the desired printer with desired magnification it is necessary to use the command

```
mfjob /a mff.mfj m=<printer-mode> s=<magn> f=<fontname>
```

(option /a forces the program to generate the output font even if it was already generated). The program *MFJob* put .tfm and .log files at the current directory, and .pk file at the subdirectory with the name which mirrors the font resolution. Example:

```
mfjob /a mff.mfj m=lj s=1 f=zcmr10
```

To teach $\text{emT}_{\text{E}}\text{X}$ to look for `.tfm` files at the current directory as well as at the system directories, it is necessary to specify the DOS environment variable `TEXTFM` as

```
SET TEXTFM=%EMTEXDIR%\TFM!;. \
```

instead of its default value `%EMTEXDIR%\TFM!` (it is assumed that `EMTEXDIR` is specified already like `SET EMTEXDIR=C:\EMTEX`).

To teach *DVI*-drivers to look for `.pk` files at the subdirectories of the current directory, it is necessary to edit the printer configuration files placed by $\text{emT}_{\text{E}}\text{X}$ at `\EMTEX\DATA\`. Suppose you use the *HP Laser Jet* printer (the modifications of the configuration files for other printers are performed similarly). The original *Laser Jet* configuration file `lj.cnf` looks like

```
% lj.cnf (300x300 DPI using LJ fonts)
+dvi-file={,$DVIDRVINPUT:}@i
+font-libraries=$DVIDRVFONTS:lj_{base,more}
+font-files=$DVIDRVFONTS:pixel.lj\@Rrdpi\@f{.pk,.pxl}
+graph-files={,$DVIDRVGRAPH:}\@Rrdpi\,@PBf{@Ef,.msp,.pcx,.bmp}
+resolution=300
+font-resolution=300
+font-scaling=1
+metafont-mode=laserjet
+max-drift=2
```

The new configuration file (with the file name *different* from `lj.cnf`) should contain the parameter `+font-files` in a form

```
....
+font-files={$DVIDRVFONTS:pixel.lj\,@Rrdpi\@f{.pk,.pxl}
....
```

Now it is necessary to substitute the references on this configuration file in the batch files `v.bat`, `vs.bat` and `prt1j.bat` from `\EMTEX\BIN\`, and you are ready to work with `MFF.STY`. Just the same re-configuration of $\text{emT}_{\text{E}}\text{X}$ will enable to work correctly with the *MFP*_{*C*} macro package.

10 History

- Ver. 0.?? — the first attempt which helped to learn more about $\text{T}_{\text{E}}\text{X}$ commands and the internal structure of *Computer Modern* fonts.
- Ver. 1.0 — the arithmetical macros finally works. The “empirical” approximation scheme based on linear splines and “corrected-by-hand” CM parameters is implemented. The fonts generated by command `\MFFmixture` (at that moment it was called `\MFFcompose`) are tested experimentally.

- Ver. 1.1 — SAUTER approximation scheme is used instead of the “empirical” approximation scheme. Commands for the logical flags are added. Font classes other than CMR are included. Generation of SMALL CAPS and *italic* fonts is added. As a result new commands are added, some commands are renamed.
- Ver. 1.2 — the font tricks described in [6] and [7] are added. Font classes corresponding to the DC FONTS driver files are included.
- Ver. 1.21a — the approximation based on DC FONTS data is added. Information printed at the header of .mf file becomes more detailed now. The MFF.STY commands are revised and the names of many commands are changed. This version was distributed during the TUG-96 Conference.
- Ver. 1.21b — the section “*Last minute corrections*” is added to this manual. The obligatory prefix *xx* is added now in front of the file name containing the font header. The list of parameters for \MFFtrick is changed so that in future the *underlining* effect can be added. This version is put on CTAN.

11 Last minute corrections

The following data is the result of discussions held during the TUG-96 Conference. Some corrections suggested by the participants are implemented just now, most of them will be implemented in future versions:

- *Add special (and obligatory) prefix in front of the file name so, that the fonts created by MFF.STY cannot be confused with the fonts created by the professional font designers.*

Corrected. Now all the fonts created by MFF.STY are started with *xx*. This feature is added to lock the attempts to create user-defined fonts under the names of the Computer Modern, etc., fonts. The prefix *xx* used by default is defined as

```
\def\MFFprefix{xx}
```

and can be redefined by the User, if necessary.

- *Include Concrete Fonts into the set of fonts used in the arithmetical mixture of Computer Modern families.*

It is not done, and with great probability it will not be done at all. The most essential effects associated with *Concrete Font Family* can be produced in MFF.STY using the scaling of the thickness for the thin strokes (see section 4.3).

- *Extend MFF.STY to generate the mathematical fonts as well.*

This is a very interesting and important suggestion, but it requires a lot of time since it is also necessary to describe all these fonts as the mathematical ones inside \TeX . Will be done someday in future.

- *Extend MFF.STY to work with Pandora Font Family as well.*

Interesting and promising. It may be done if there is enough time to look for for *Pandora Font Family* in more details.

- *Generate the partial fonts, i.e., generate the font where some (or nearly all) characters from the original MetaFont source are excluded.*

Interesting and not too difficult using the re-definition of the procedure `beginchar` by the commands inside the font header created by MFF.STY. Some problems may arise due to *Metafont* memory problems. Will be done in future.

- *Include some more font trick effect, namely the underlining of the characters so that the underlining stroke has some gaps near the descender of the character.*

Interesting and not too difficult. Corresponding feature is reserved now in the list of `\MFFtrick` parameters (see section 7). Will be realized in the nearest future.

Acknowledgements

This research was partially supported by a grant from the Dutch Organization for Scientific Research (NWO grant No 07-30-007).

References

- [1] Donald E. Knuth. *Computer Modern Typefaces (Computers & Typesetting series)*. Addison-pt Wesley, 1986.
- [2] John Sauter. *Building Computer Modern fonts. TUGboat*, **7** (1986), pp. 151–152.
- [3] Jörg Knappen. *The release 1.2 of the Cork encoded DC fonts and the text companion symbol fonts*. Proceedings of the 9th Euro \TeX Conference, Arnhem, 1995.
- [4] A.Khodulev, I.Mahovaya. *On \TeX experience in MIR Publishers*. Proceedings of the 7th Euro \TeX Conference, Prague, 1992.

- [5] O.Lapko. *MAKEFONT as a part of CurTUG-EmTeX package*. Proceedings of the 8th EuroTeX Conference, Gdańsk, 1994.
- [6] Georgia K.M. Tobin. *The ABS's of Special Effects*. TUGBoat **9** (1988) No 1 pp. 15–18.
- [7] Doug Henderson. *Outline Fonts with Metafont*. TUGBoat **10** (1989) No 1 pp. 36–38.