VFComb 1.3 — the program which simplifies the virtual font management

A.S. Berdnikov
Institute of Analytical Instrumentation
Rizsky pr. 26, 198103 St.Petersburg, Russia
berd@ianin.spb.su

S.B. Turtia
Institute of Analytical Instrumentation
Rizsky pr. 26, 198103 St.Petersburg, Russia
turtia@ianin.spb.su

The MS DOS program *VFComb* enables one to simplify the design of the virtual fonts.[1][1] Its main purpose was to facilitate the integration of CM fonts with Cyrillic LL fonts created by O. Lapko and A. Khodulev [2, 3] but it can be used for other applications too. It uses the information from `.tfm` files (converted to ASCII form by *TFtoPL*) and the ASCII data files created by the User on its input, and produces the `.vpl` file on its output (the `.vpl` file can be converted later to the virtual font using *VPtoVF*). The characteristic feature of the program is that it can assemble the ligature tables and metric information from various fonts and combine it with the user-defined metric information and ligature/kerning data. *VFComb* supports the full syntax of `.pl` files and `.vpl` files as it was defined by D.E. Knuth and adds new commands like symbolic variables or conditional operators, which simplifies the creation and the debugging of the virtual fonts.

The description of the previous version 1.2 (which is the first version distributed far outside the home computer) can be found in [4]. This version has only the Russian manual which prevents its wide distribution among TEX community. The current version *has* the English manual, but except this "new feature" a lot of additional improvements are added. The main features of the program are described below while the complete information can be found inside the manual.

The program will be put on the CTAN archives following *TUG'96* together with the source code and will be available to TEX community on a freeware basis. Since this MS DOS program is written on *Borland Pascal* and uses some specific features of this language, it is hardly portable to any other platform "as is", but it is not too difficult to transfer it to portable ANSI C (volunteers are welcome).

[1] It is assumed that you are familiar with the virtual fonts. If it is not so, it is highly recommended that you read [1] before proceeding further.

Virtual fonts for TEX formats with national alphabets

Although everything which can be done by *VFComb* could be realized also by explicit usage of PL and VPL file syntax (as well as everything which can be done manually by `.pl` and `.vpl` files can be done with *VFComb*), some typical operations with the virtual fonts are performed with its help *easier* than by manual editing of `.pl` and `.vpl` The typical problem of this type is the adaptation of standard TEX formats to national alphabets — this problem is especially important for Cyrillic alphabets since most Cyrillic letters *cannot* be created as the combination of the Latin (English) letters with some accents.

The standard solution of this problem is to combine the English part taken from Computer Modern family with the national fonts which extend the Computer Modern family and which contain in the upper part of ASCII table (codes 128 – 255) the national symbols. The best way how to do it is to create the *virtual font* whose lower part refers to original CM fonts, and upper part refers to the national fonts — it is just the way which was recommended by D. Knuth.[1] The advantage of this approach is that it is possible to keep the changes in CM fonts and in national fonts separately, and in addition, it is possible to economize disk space since it is not necessary to keep *two* copies of each Computer Modern character — one as the original CM font which is necessary for original TEX formats, and the second one as the lower part in the combined national font.

The combination of the lower part of one font and the upper part of another font, or even the joining all the characters from one font and all the characters from another font (provided that *no* character code is encountered twice) can be done by *VFComb* using few commands. If some characters are to be discarded from the font or moved to the different positions of the ASCII table, it does not makes the command script more complicated. The resulting virtual font contains proper metric information for each character borrowed from the source metric information and the correct mapping of the characters into individual real fonts.

The similar operation can be performed also by the program *TFMerge* (IHEP TEXware, Protvino), but *VFComb* performs additional operations. That is, in addition to joining metric and ligature/kerning information from each font into one virtual font, it is necessary to add *cross*-ligature and *cross*-kerning information for the pairs of characters taken from different fonts. *VFComb* enables one to add metric, ligature and kerning data taken from its script file (which makes the original script a little bit more complicated). The important feature is that this additional data can contain *variables* and *logical structures*, from which one can generate the whole CM family of the virtual typefaces with national characters using just the same pseudo-program written on *VFComb* command language.

Except the operations described above, *VFComb* is capable of performing the following operations if it is specified by the user in its script:

- discard the ligature tables of some real fonts;
- include in the virtual font the full ligature table of the real font;
- include in the virtual font only those characters which are declared explicitly in user-defined data, and discard the elements of the ligature tables which correspond to non-included characters of the real font;
- automatically add to the virtual fonts the characters which are not included explicitly by the user but which are joined with the already included characters through ligature table data, or by specifications `NEXTLARGER` and `VARCHAR`.

These features allow creation of the desired virtual fonts for national alphabets with less effort and with more reliability than by manual manipulations with `.pl` and `.vpl` files.

Virtual fonts for colored printing

Another problem is the application of the virtual fonts to multicolored printing. Suppose that it is necessary to print text where different characters have different colors. From TEX-compiler's point of view it means that the characters with different colors are assigned to different fonts, and it is a task for the `dvi` driver to decide how to print these fonts in desired colors.

The colored printing is collected from the overlapped sheets where each sheet of text or graphics is printed by individual monocolor passes. To make the templates for monocolor printing it is necessary to organize the output of the `.dvi` file so that in one pass only yellow characters are printed, in another pass only blue characters are printed, etc., while the characters which have the green color are to be printed twice — in blue as well as in yellow. The easiest way to teach `dvi` driver how to do it is to create different subdirectories with virtual fonts — one subdirectory for each elementary color. The virtual font files placed in the subdirectory for yellow printing (which corresponds to the yellow fonts) will refer to the actual `*.pk` files if and only if the yellow color is assigned to this character — otherwise it will refer to *empty* character. The subdirectories for other colors are organized similarly. As soon as the yellow printing is performed, the `dvi` driver is configured so that it takes the virtual fonts from the "yellow" subdirectory, and for the output in other colors a corresponding reconfiguration of the `dvi` driver is performed.

If the mapping of the empty characters into the `dummy` font is performed, it results to the wrong behaviour of the `dvi` driver: the characters in the `dummy` font have zero size, and this means that the next character after the empty character is shifted to the left (as compared with the desired behaviour) a the distance equal to the width of the skipped character. To prevent this effect it is necessary to insert into the virtual font the explicit `dvi` commands which move the current output position to the right by the distance representing the width of the skipped character. This operation is performed by *VFComb* by a single command: the user assigns the attribute `NULLCHARACTER` to the corresponding real font, and for the characters of this font the *empty* mapping will be performed instead of mapping the real font characters.

Substitution of CM fonts instead of PostScript fonts for *DVI* Viewers

The next problem where the usage of the virtual fonts is advantageous is the visualization of the document which was compiled using PostScript fonts. Generally, the screen viewer *cannot* process the PostScript characters, and it is necessary to remap the PostScript font characters into some `.pk` font which *can* be displayed by the viewer — say, some typefaces from the Computer Modern family.

Such remapping can be performed using virtual font mechanism, but if it is done without special precautions the screen view can be far from the printed output. The reason is that CM characters have a width different from the PostScript font (the fact that they have a different graphical image is not so essential). As in the previous case, the screen output will be shifted to the left on the distance which is the difference between the width of the PostScript character and the CM character if no special precautions are taken. To make the correct output, it is necessary to add to the virtual font the explicit `dvi` commands which correct the current output position.

To make corresponding virtual font automatically, *VFComb* allows the user to specify for the real fonts *two* `.pl` files with the metric information: the first one is for the nominal characters which are used by TEX to compile the `.dvi` file (in our case it is the PostScript `.afm` file converted to `.tfm` format), and the second one is for the real characters which are used when the `.dvi` file is displayed (or printed). If such information is specified by the user, the commands to correct properly the current output position are inserted into the virtual font.

This operation works if both fonts have the same coding scheme – namely, the characters used by TEX and the characters used by `dvi` viewer have the same code value. If not, the operation of remapping inside an already-mapped font is required, and this could be very complicated and result in a very complicated scheme of virtual font generation. To solve this problem, it is assumed that the correct metric information for the "true" font (i.e., for the font used in compilation of the `.dvi` file), is already available. The special operators in *VFComb* enable the user to load this information and to correct the proper character width.

Other features

The other features incorporated in *VFComb* 1.3 are:

- improved syntax for *VFComb* commands;
- improved logical operators:
- implementation of string variables in *VFComb* script files;
- specification of variable values in command line among other parameters;
- specification of the names of the real and virtual fonts inside *VFComb* scripts instead of command line;
- correction of some bugs, including the obligatory conversion to uppercase all input characters together with the font names;
- automatic computation of the metric information for the characters composed from user-defined `dvi` commands.

All of these improvements ensure easier use of the program. For example, after implementation of the new features, the generation of the virtual fonts for the LL/LH family (used in the CyrTUG Cyrillic version of TEX/LATEX/$\mathcal{AMS}$-TEX) is performed using just *one* script file of *VFComb* instead multiple header files (see the example below).

Comparison with *FontInst*

There is another package for manipulating with virtual fonts — namely, *FontInst* by Alan Jeffrey. Although there are many similar features between *FontInst* and *VFComb*, these tools are designed to solve different tasks.

The main purpose of *FontInst* is to create new font families for LATEX 2$_\varepsilon$ using existing PostScript fonts. *FontInst* contains high-level operators which enable one to perform this task in several commands, and it is written totally in TEX, which guarantees its high portability. In addition, *FontInst* contains special TEX macro commands which enable the user to do nearly anything with virtual fonts, without direct editing of `.vpl` files; but, in this case the "program" written in *FontInst* commands may be comparable in length to the `.vpl` file.

VFComb is designad to solve different problems — it was designed mainly to simplify the integration of new alphabets into TEX so that the Latin part of Computer Modern Typefaces is left unchanged. It *cannot* read `.afm` files and it *cannot* create `.fd` files. If you wish to use it to create virtual fonts for PostScript, you can do so, but you need some additional utilities (`afm2tfm`) and many more manual operations than you need with *FontInst*.

Similarly, you can make virtual fonts like that created by *VFComb* using *FontInst* as well, but this will require manual manipulations which are comparable to the direct editing of `.vpl` files. Although one can expect in future more convergence between *FontInst* and *VFComb*, currently these programs do not intersect in this respect. If you wish to create virtual fonts for font families derived from PostScript, please use *FontInst*, and you will economize a lot of your time. If you wish to make virtual fonts which solves any of the problems described in this paper, *VFComb* may be more suitable.

Example

The syntax of *VFComb* commands is similar to that of `.vpl` files. It means that it is more suitable for computers than for people. The following example is extracted from the file `lhfonts.tbf` used to generate the full family of Cyrillic virtual fonts demonstrates the *VFComb* syntax.

Suppose that the program *VFComb* is called as

```
vfcomb lhfonts.tbf font=lhr10
```

which means that the script file `lhfonts.tbf` is used as a source of *VFComb* commands, and that the user asks to generate the virtual font `lhr10` which is a combination of the *Latin part* taken from `cmr10` and the *Cyrillic part* taken from `llr10`. The fact that the virtual font has the name `lhr10` and that it is composed from `cmr10` and `llr10` is described inside the script file `lhfonts.tbf`. Actually the command line shown above specifies only the fact that the string variable `font` is defined with the initial value `lhr10`, before the script file `lhfonts.tbf` is processed.

The head of the file `lhfonts.tbf` contains the commands

```
(IF-DEF font)
    (VARIABLE (STRING FONT @V font))
(ELSE)
    (MESSAGE Variable FONT is not defined)
    (HALTPGM)
(ENDIF)
```

These commands check that the user does not forget to specify the variable `font=...` at the command line, and assign its value to the string variable `FONT` (uppercase and lowercase letters are distinguished by variable names, and the prefix `@V` means the value of the string variable).

Similarly, the commands

```
(IF-DEF type)
     (VARIABLE (STRING TYPE @V type))
(ELSE)
     (VARIABLE (STRING TYPE NEW))
     (MESSAGE TYPE = NEW is assumed)
(ENDIF)
```

analyzes the contents of the variable `type` if it is specified at the command line, and assigns the default value `NEW` if there is no expression `type=...` at the command line.

The commands

```
(VTITLE CyrTUG freeware LH font family)
(OUTPUT @V FONT)
(HEADER (FONT D 1))
(MAPFONT D 0 (LOWPART))
(MAPFONT D 1 (HIGHPART))
```

specify the title of the virtual font and the name of the output virtual font. The *header* of the virtual font is similar to that of the font `D 1` if there are no other commands which modify the contents of the header. The commands `MAPFONT` state that two fonts are used to create the virtual fonts where the characters 0 – 127 are taken from one font and the characters 128 – 255 are taken from the other font.

The actual names of the real fonts '`0`' and '`1`' are specified after the following analysis of the contents of the variable `FONT`:

```
  (IFS-CASE @F @V FONT)
  (CASE LHR)
     (MAPFONT D 0 (FONTNAME @+ CMR @P @V FONT))
     (MAPFONT D 1 (FONTNAME @+ LLR @P @V FONT))
     (HEADER (FAMILY LHR))
     (VARIABLE (BYTE FLKERN 0))
     (BREAK)
  (CASE LHTI)
    (MAPFONT D 0 (FONTNAME @+ CMTI @P @V FONT))
    (MAPFONT D 1 (FONTNAME @+ LLTI @P @V FONT))
    (HEADER (FAMILY LHTI))
    (VARIABLE (BYTE FLKERN 1))
    (BREAK)
......
(ELSE)
   (MESSAGE Unknown font family @F @V FONT)
   (HALTPGM)
(ENDIF)
```

Here the prefix commands `@F @V FONT` extract the non-digital component of the font name which is analysed by the `CASE` operators (note that the comparison of text strings by the operator `IFS-CASE` does not distinguish between uppercase and lowercase letters). If the non-digital font component is equal to `LHR` (as it is in our case for `font=lhr10`), the first font gets the name `CMR...`, and the second font gets the name `LLR...` where the dots are substituted by the font design size: the prefix `@P @V FONT` extracts the value '10' from `lhr10` (like `@F @V FONT` extracts '`lhr`' from the same value), and the prefix `@+` performs the concatenation of two strings. In the same command block, the header field `FAMILY` gets the value `LHR`, and the byte variable `FLKERN` gets the value 0 (it is used later to construct the additional entries for ligature table which corresponds to the ligature/kerning data for the characters taken from different real fonts). The font names `LHTI...` are analyzed similarly, and analogous commands (skipped here) are specified for every legal font family.

The following commands specify the symbolic names for some Cyrillic letters which are used later in the ligature tables. Note the the coding scheme specified here depends on the value of the expression `type=...` specified in the command line:

```
  (IFS-EQ @V TYPE OLD)
     (VARIABLE
       (BYTE CYR_open_quote  D 243)
       (BYTE CYR_close_quote D 244)
```

```
      (BYTE CYR_Number      D 245)
    )
  (ELSE) (COMMENT TYPE = NEW)
     (VARIABLE
        (BYTE CYR_open_quote  D 250)
        (BYTE CYR_close_quote D 251)
        (BYTE CYR_Number      D 252)
     )
  (ENDIF)
  (VARIABLE
    (BYTE CYR_GHE D 131)
    (BYTE CYR_ghe D 163)
    (BYTE CYR_ER  D 144)
    (BYTE CYR_er  D 224)
    ......
  )
```

The following commands analyze the value of the variable FONT and assign the value for the real variable u# and for the logical variable monospace, which are used to construct the ligature data for the pairs of characters taken from different fonts:

```
  (IFS-CASE @V FONT)
  (CASE LHR5)
      (COMMENT Font LHR5)
      (VAR
         (REAL u# A/ A/ R 12.5 R 36 R 5)
         (BYTE monospace D 0)
      )
      (BREAK)
  (CASE LHR6)
      (COMMENT Font LHR6)
      (VAR
         (REAL u# A/ A/ R 14 R 36 R 6)
         (BYTE monospace D 0)
      )
      (BREAK)
  ...
  (ENDIF)
```

The arithmetic expressions in these commands are constructed using the "*Polish notation*" structure. That is, the prefix A/ is the division of the two arguments which follow it (A+ is addition, A- is subtraction, A* is multiplication and R is a real number), and each of these arguments can be the arithmetic expression starting with an arithmetic prefix as well. For example, the value of the variable u# for the font LHR5 is equal to A/ A/ R 12.5 R 36 R 5 $= ((12.5/36)/5) = 0.069444444444$.

Finally, the commands which specify the additional ligature data for the pairs of characters taken from different fonts are added:

```
  (IF-EQ V monospace D 0)
  (IF-CASE V FLKERN)
  (CASE D 0) (COMMENT KERN Roman)
     (VARIABLE
        (REAL k#   A* R -0.5 V u#)
        (REAL kk#  A* R -1.5 V u#)
        (REAL kkk# A* R -2.0 V u#)
        ......
     )
     (LIGTABLE
        (LABEL V CYR_GHE)
        (KRN C . V kk#)
        (KRN C , V kk#)
        (KRN C : V kk#)
        (KRN C ; V kk#)
        (STOP)
     )
     (LIGTABLE
        (LABEL V CYR_ER)
        (KRN C . V kk#)
        .....
     )
     .....
     (BREAK)

  (CASE D 1) (COMMENT KERN Italic)
     ......
  (ENDIF)
  (ENDIF)
```

The syntax of these commands is similar to that of .vpl files except that the variable values are used (some variables are calculated depending on the variable u# defined above) and the logical operators analyze what data should be included depending on the current values of the variables FLKERN and monospace.

Acknowledgements

All new improvements of *VFComb* (except the English manual) are the result of the contacts and discussions which were held during the *EuroTEX-95* meeting. So we would like to thank Kees van der Laan for his giant efforts to organise the visit to the *EuroTEX-95* by the delegation from Russia and for his patient attention to Russian colleagues before, during and after the *EuroTEX-95*.

It is not so easy to recall *all* participants of this conference whose opinions made an impact on the preparation of the new version of *VFComb*. Among other persons we would like to thank Phil Taylor and S. Znamensky for their valuable suggestions which led to improvements in the program. We would like also to thank O.A. Lapko, S.A. Strelkov and I.A. Makhovaya for their efforts spent on the Cyrillic TEX project which actually inspired our work.

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

References

[1] D. Knuth, "Virtual Fonts: More Fun for Grand

Wizards", TUGBoat **11** (1993), No. 1, pp.13–23.

[2] A. Khodulev, I. Mahovaya. "On TEX experience of MIR Publishers", Proceedings of the 7th EuroTEX Conference, Prague, 1992.

[3] O. Lapko. "MAKEFONT as a part of CyrTUG–EmTEX package", Proceedings of the 8th EuroTEX Conference, Gdańsk, 1994.

[4] A.S. Berdnikov, S.B. Turtia. "*VFComb* — a program for design of virtual fonts", Proceedings of the 9th EuroTEX Conference, Arnhem, 1995.