VFComb 1.3

the program for the designers of the virtual fonts

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

June 22, 1996

Introduction

The MS DOS program *VFComb* enables to design the virtual fonts for TEX, LATEX, etc., using already existing real or virtual fonts [1][1] in a more flexible manner than the direct manipulations with `.vpl` files. It simplifies the process of combining real fonts into the virtual font and makes the debugging of the virtual fonts more simple and less time consuming. The main purpose of this program 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[2].

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. The program uses the information from `.tfm` files (converted to ASCII format by the utility *TFtoPL*) and the ASCII data files created by the User, and produces the ASCII `.vpl` file on its output. The `.vpl` file can be converted later to the virtual font using the utility *VPtoVF*.

VFComb supports the full syntaxis of `.pl` and `.vpl` files as it was defined by D.E.Knuth [1] and adds new commands like numerical variables, pseudo-arithmetics operations and conditional operators, which simplifies the design of the virtual fonts. The typical virtual font operations are performed automatically and require a compact *VFComb* script file. The *VFComb* script files have flexible structure so that just the same script files can be used to create the full generic font family.

The commands from the *VFComb* script file enables:

- specify the information about the mapping table between the characters of the virtual font and the real fonts;
- specify the explicit sequences of *DVI*-commands substituted instead of some characters of the virtual font;
- extract metric information and ligature tables from the `.pl` files (converted from `.tfm` files) which describe the properties of the real fonts;
- add user-defined metric information and ligature tables (for example, the ligature and kerning data for the pairs of characters corresponding to different real fonts);
- include automatically new characters if these characters are connected by the attributes `VARCHAR`, `NEXTLARGER` or ligature table data with the characters already included into the virtual font;

[1] It is assumed that you are familiar with the concept of the virtual fonts and with the article [1]. If it is not so, it is highly recommended to read [1] before you proceed further.

[2] Since this MS DOS program is written on *Borland Pascal* and uses some specific features of this language, it is hardly portable "as it is" to any other platform, but to my mind it is not too difficult to transfer it to portable ANSI C (volunteers are welcome).

- insert automatically the DVI-commands which correct the DVI-cursor position if the metric information from the virtual font is different from that for the real font;
- create the output .vpl file which combines all these data.

All input and output files are the ordinary ASCII files. The command line of the program specifies the name of the script file (the script file contains the commands which control the work of the program), and, may be, defines some variables which are analyzed and used later inside the script file. The output of the program is the .vpl file which contains all information necessary to create the virtual font using the utility *VPtoVF* and the .log file which summarizes all printed messages and all operations performed by *VFComb*.

1 Command-line parameters

The command line contains the list of parameters which are separated by spaces (the body of the parameter should contain *no* spaces). The following parameters can be specified:

- the name of the script file (the obligatory parameter);
- the definition of some variables which are used later in the script file (variable specifications always contain the character '=' in its body);
- the name of the log-file, the names of the directories which used as a source for the input files, the directory for the output files, etc. (these parameters start with the character '/'[3]).

The order of the parameters is of no sence[4]. The *responce files* (see section 1.4) specified in a form '@*filename*' and DOS environment variable VFCOMB (see section 1.5) can be used to define the parameters which are too long or which are specified too often to be typed manually at the command line.

1.1 Script file name

The parameters which specify the name of the log-file, the names of the directories which are used as a source for the input files, the directory for the output files, etc., start with the character '/' (the characters '!', '-', '+' can be used as

[3]The characters '!', '-', '+' can be used as well.

[4]The *only* exception: if the .log file name specified by the command line parameter /o does not contain the explicit directory specification, the directory specified by the parameter /o is used only if this parameter is specified *before* the parameter /l (the command line parameters /l and /o are described in section 1.3).

well). The values of variables specified at the command line contain the character '=' in its body. The *responce file* name starts with the character '@'. All other constructions are recognized as the *script file* name.

The *script file* is the main source of the commands which are processed by *VFComb* and which control the creation of the output `.vpl` file. If the script file name has no extension, the extension `.tbf` is added automatically. If the script file name has no explicit directory specification and it is not found in the current directory, *VFComb* looks for it in the directories specified by the parameter `/u`.

The User should specify only *one* script file at the command line (previous versions of *VFComb* enabled to specify *several* script files in the command line). If several script files are specified (or if there are the command line parameters which are not recognized by *VFComb* properly as the variable specifications, responce file names, etc.), the new name redefines the previous one after the warning message. If one needs to combine several `.tbf` files into one stream of commands, the command `LOADFILE` (section 2.1.5) specified inside the script file can be used to do it.

1.2 Variables

The script file commands can use the variables to specify the numerical and text data. Usually the variables and their values are specified by the script file commands described in section 2.1.3. This section considers how it is possible to specify the variables and their values *at the command line* so that they can be analyzed by the script file commands when the script file is processed by *VFComb*.

The definition of the *string* variable and its initial value is performed by the construction "*name*=*value*" — when such parameter is encountered in the command line, a new string variable *name* is allocated, and the text string *value* is assigned to it.

The *numerical* variables can be specified if the special suffix is added at the end of the field *value*[5]:

- `/r` or `/R` corresponds to *real variable*;
- `/F`, `/D`, `/H`, `/O`, `/B`, `/C` correspond to *32-bit integer* variable where `/F` means *face code* value, `/D` means *decimal* value, `/H` means *hexadecimal* value, `/O` means *octal* value, `/B` means *binary* value, `/C` means *character code* (integer value);
- `/f`, `/d`, `/h`, `/o`, `/b`, `/c` correspond to the *byte* (8-bit integer) variable which are usually used to define the character codes — similarly to integers, `/f`

[5] Usually it is not necessary since the User can allocate the numerical variables and assign the numerical values to them just by analyzing the contents of the string variables specified at the command line.

means *face code*, `/d` means *decimal* value, `/h` means *hexadecimal* value, `/o` means *octal* value, `/b` means *binary* value, and `/c` means *character code*;

- `/@F`, `/@D`, `/@H`, `/@O`, `/@B`, `/@C` and their lowercase forms define the string variable with the contents equal to decimal representation of corresponding integer value — for example, the parameter '`tmp=377/@O`' is equivalent to '`tmp=255`' (octal number 377 has the decimal representation 255). The suffix '`@`' is equivalent to '`@D`' and '`@d`', and all three specifications are of no use since '`S=12345/@D`' is just the same as '`S=12345`'.

The contents of the field *value* is to be in agreement with the suffix specification (see section 2.1.1 for more details). Examples:

`vfcomb fntnam=CMR10 ...` — string variable;

`vfcomb rval=3.14159/r ...` — real variable;

`vfcomb ival=CDEF/H ...` — integer variable with hexadecimal value;

`vfcomb cval=X/c ...` — byte variable which is the character code for `"X"`.

1.3 Other parameters

The parameters which specify the log-file name, etc., start with one of the characters '`/`', '`!`', '`-`', '`+`' followed by the *parameter identifier* (a single character) and the parameter value. The parameter identifier and the parameter value can be separated by '`:`' or '`=`' to achieve better readability. Example:

```
vfcomb ...  /l:vfcomb.log -u=c:\myinput;c:\ !oOUTPUT\ ...
```

The previous version of *VFComb* used a lot of such parameters but now most of them are moved inside the script file. The following parameters can be specified at the command line:

- `/u=`*list-of-dirs* — specifies the list of directories separated by '`;`' which are used to search for the input files if the input file is absent in the currect directory. Several parameters `/u` can be specified — in this case they are joined together. The maximum number of the search directories is 20, and the new emTEX directory search algorithm (see emTEX documentation) is not implemented.

- `/l=`*log-file-name* — specifies the name for the `.log` file which summarizes the operations performed by *VFComb*. By default the name *scriptfile*`.LOG` is used, but if the script file is unknown when the first *VFComb* message is printed, the `.log` file has the name `VFCOMB.LOG`.

- `/o=`*output-dir* — specifies the directory where the output files (`.vpl` and `.log` files) are to be placed if their directories are not specified explicitly.

1.4 Responce files

The User can specify the construction '`@`*filename*' among the list of the command line parameters which means that the contents of the file *filename* is added to the contents the command line.

Each line of the responce file is interpreted as a separate parameter. The initial spaces of the line are omitted, and the characters after the first space which terminates the parameter are omitted too. The lines which start from the characters `'%'`, `';'`, `'*'` and the empty lines are ignored.

The responce files can be nested in each other (that is, the responce file can have the line `@`*filename*1 again) but the depth of their nesting cannot be greater than 5.

1.5 Environment variable `VFCOMB`

The User can define the DOS environment variable `VFCOMB` using the command

```
SET VFCOMB=......
```

The contents of this variable is added *at the beginning* of the command line. As a result, the contents of the command line redefines the default specifications established by the environment variable `VFCOMB` if some parameters are specified twice.

2 Script file commands

The *VFComb* syntaxis is very close to the syntaxis of `.vpl` files (it enables to economize the program code since *VFComb* is to read and to analyze the `.vpl` files in any case). That is, the `.tbf`, `.pl` and `.vpl` files are composed from the entries specified in a form

```
(PROPERTYNAME VALUE)
```

where the `VALUE` can contain nested list structures as well.

The following extensions of `.pl` and `.vpl` syntaxis [1] are used by *VFComb*:

- new types of numerical constants;
- numerical variables;
- pseudo-arithmetical expressions;
- string variables and string expressions;
- conditional operators.

These commands are described in details in section 2.1.

The commands specific to *VFComb* `.tbf` files enable to specify the following data:

- the name of the virtual font (actually the name of the output `.vpl` file);
- the header data which is specified at the beginning of the virtual font;
- the assignment (if necessary) the numerical values to symbolic variables used in specification of other user defined data;
- the table(s) for mapping the characters of the real fonts into the characters of the virtual font;
- the additional ligature table(s) (if present) which contain the ligature and kerning information for the pairs of characters corresponding to different real fonts;
- the metric information, ligature and kerning tables, etc., for the characters of the real fonts;
- the options which control the process of creation of the virtual font.

2.1 Extensions of `.vpl` files

The following commands and extensions of numerical data specifications are valid for *VFComb* script files (`.tbf` files) and for `.pl` and `.vpl` files as well. These commands are processed inside *VFComb* "mouth" subroutines, and as a result they are invisible for the subroutines which analyze the syntaxis of `.tbf`, `.pl` and `.vpl` files.

The structure of the algorithm which analyzes the syntaxis of the input commands implies some restrictions on the way these *preprocessor* commands can be used in the source files. The *preprocessor* commands can be inserted at any point where the argument `(......)` is suitable. So, they can be specified on the upper level of braces, they can be specified as the argument of the command `(CHARACTER ...)` since it *has* the sub-list arguments, but they cannot be specified inside the command `(CHECKSUM ...)` since there are *no* sub-list agruments for this command.

2.1.1 Numerical values

The syntaxis of `.vpl` files enables to use the numerical data of different forms. The numerical data can be specified as real (floating point) values, 32-bit integer values and 8-bit integer values. Depending of the context, real or integer values should be used, and the range if the accepted integer values depends on

context also (say, character code cannot be greater than 255 while the `checksum` parameter is a full 32-bit integer).

Each value starts with the prefix letter preceeded by a space and separated from the following actual value by a space (as usually, there may be as many spaces as you need at any place where at least one space is allowed). The prefix character defines the type of the numerical value, and the following types can be used:

`R` *value* — real values (`R 0.27182`);

`D` *value* — decimal integer values (`D 123456`);

`O` *value* — octal integer values (`O 377`);

`H` *value* — hexadecimal integer value (`H 07FA`);

`C` *character* — character numerical code (`C x` is the code value for character '`x`');

`F` *ident* — Xerox *face code*, i.e., an integer value which is substituted by a 3-letter identifier (`F LIE` = `D 17`) where the full list of the face codes assigned to the integers from 0 to 17 is: MRR, MIR, BRR, BIR, LRR, LIR, MRC, MIC, BRC, BIC, LRC, LIC, MRE, MIE, BRE, BIE, LRE, LIE.

For *VFComb* input files in all places where the real values can be used, the integer values can be used also, and in all places where 32-bit integers are accepted, 8-bit integers are accepted too. *VFComb* also extends the set of prefix letters so that more general and more flexible expressions can be constructed:

`B` *value* — binary integer value (`B 011111111010`);

`V` *var-name* — the value of some numerical (real or integer) variable (`V FntSize`);

`V-` *var-name* — the *negated* value of some numerical (real or integer) variable (`V- FntSize`);

`V+` *var-name* (unary '+') — just the same as `V` *var-name* (`V+ FntSize`);

`V~` *var-name* — bit-by-bit *negated* value of an integer variable (`V~ FntSize`);

`A+` *value1 value2* — addition of two (real or integer) values ('`A+ D 2 D 2`' = 4);

`A-` *value1 value2* — subtraction of two (real or integer) values ('`A- D 5 D 3`' = 2);

`A*` *value1 value2* — multiplication of two (real or integer) values ('`A* D 5 D 3`' = 8);

`A/` *value1 value2* — division (real or integer) of two (real or integer) values (‘`A/ D 5 D 2`’ = 2 for integers, ‘`A/ R 5.0 R 2.0`’ = 2.5 for real values);

`A%` *value1 value2* — the residue of integer division (mod) of two integer or real values (‘`A% R 5.0 R 2.0`’ = 1.0);

`A!` *value1 value2* — bit-by-bit **or** operation between two integer values (‘`A! B 1010 B 0011`’ = `B 1011` = 11);

`A&` *value1 value2* — bit-by-bit **and** operation between two integer values (‘`A& B 1010 B 0011`’ = `B 0010` = 2);

`A^` *value1 value2* — bit-by-bit **xor** operation between two integer values (‘`A^ B 1010 B 0011`’ = `B 1001` = 9);

`A>` *value1 value2* — right shift of integer *value1* by *value2* bits (‘`A> B 1010 D 2`’ = `B 10` = 2);

`A<` *value1 value2* — left shift of integer *value1* by *value2* bits (‘`A< B 1010 D 2`’ = `B 101000` = 40);

`-A` *value* — unary ‘−’ (‘`-A D 5`’ = -5, ‘`-A D -5`’ = 5);

`+A` *value* — unary ‘+’ (‘`+A` *value*’ = *value*);

`~A` *value* — bit-to-bit negated integer *value* (‘`~A B 1010`’ = `B 0101` = 5).

The arguments of the arithmetic expressions can be the arithmetic expressions again which enables to build complex arithmetic forms (similar to *polish notation*) — unfortunately these forms are not suitable for fast human recognition. The arithmetic operations `A!`, `A&`, `A^`, `A>`, `A<`, `~A`, `V~` can be used for integer arguments and only at the places where the integer expression is required.

Although *VFComb* syntaxis enables to use complex string constructions (see section 2.1.2), the values specified as the arguments of the prefix commands `R`, `D`, `O`, `F`, `C`, `H`, `B`, `V` are always the primitive text strings (i.e., the sequences of characters terminated by a space or the closing braces ‘`)`’). It means that the expression ‘`D @V ABC`’ where `ABC` is the string variable with the value `"99"` is illegal although one can assume that it is to be equivalent to ‘`D 99`’.

There are some serious reasons not to allow the string expressions to be the arguments of the arithmetic constants. The simplest one is, for example, that in this case it is impossible to convert the *legal* expression `C @` into numerical value `D 64` as it is required by the rules of `.vpl` files. Nevertheless, to enable to use such possibilities if necessary, the following prefix commands are added which enable to perform the conversion of the string expressions into numerical values:

`D@` *string* — interprete string expression as a decimal integer value;

`O@` *string* — interprete string expression as an octal integer value;

`H@` *string* — interprete string expression as a hexadecimal integer value;

`B@` *string* — interprete string expression as a binary integer value;

`F@` *string* — interprete string expression as an integer *face code*;

`C@` *string* — interprete string expression as a *character code* (string is to be a single character);

`R@` *string* — interprete string expression as a real value;

`V@` *string* — interprete string expression as a name of a variable (real or integer).

2.1.2 String values

Some `.pl` and `.vpl` commands require text strings as their arguments (commands `CODINGSCHEME`, `FAMILY`, `VTITLE`, for example). *VFComb* extends the set of numerical values so that the string values, string variables and string expressions can be used together with the numerical ones. The specification of the *string variables* and *string expressions* makes the specification of some commands more flexible and enables to prepare efficient and compact *VFComb* scripts.

The string constants (contrary to the numerical constants) have no special prefix — they are just the sequences of characters which can be surrounded by spaces. Contrary to that the string expressions start with the prefix '`@`' — which means that a string constant *cannot* starts from this letter but it does not restrict the flexibility in specification of the string constants too much.

The following string operations can be used:

`@V` *var-name* — the value of some string variable defined previously (`@V HlpString`) (essential note: the argument of the prefix command `@V` *can be* the string expression itself like that of the prefix command `V@` described at the previous section);

`@R` *real-value* — the real numerical value converted to a text string ('`@R A/ R 1 R 7`' = `"0.14285714286"` since '`A/ R 1 R 7`' = 1/7);

`@D` *integer-value* — the integer numerical value converted to decimal text representation ('`@D O 377`' = `"255"`);

`@O` *integer-value* — the integer numerical value converted to octal text representation ('`@O D 255`' = `"377"`);

`@H` *integer-value* — the integer numerical value converted to hexadecimal text representation ('`@H D 255`' = `"FF"`);

`@B` *integer-value* — the integer numerical value converted to binary text representation (‘`@H D 255`’ = `"11111111"`);

`@F` *string-value* — extract font name from the string (‘`@F cmr10`’ = `"cmr"`);

`@P` *string-value* — extract the string part after the font name from the string which for legal font names is the font design size in `pt` (‘`@P cmr10`’ = `"10"`);

`@U` *string-value* — convert the string value to uppercase form (‘`@U cMr10`’ = `"CMR10"`);

`@L` *string-value* — convert the string value to lowercase form (‘`@L cMr10`’ = `"cmr10"`);

`@+` *string1 string2* — combines two strings into one string (‘`@+ CMR 10`’ = `"CMR10"`);

`@` *integer-value* — string which consists from a single character with a specified code (*integer-value* is in a range 0–255);

`@@` *integer-value* — string of the specified length composed from spaces;

`@.` — empty string (just the same as `@@ D 0`).

As usually the numerical and string values can be the complex expressions, not only the elementary constants.

The sequences of string constants and string expressions separated by spaces can be specified as the argument of the commands like `FAMILY`, `CODINGSCHEME`, `VTITLE`. In this case the *VFComb* inserts a space between individual string element and expands string expressions to a form of a string constants. As a result the string composed from a sequence of words separated by spaces (where multiple subsequent spaces are compressed into one space) can be constructed and written into the output `.vpl` file.

2.1.3 Variables

As it was mentioned in sections 2.1.1 and 2.1.2, it is possible to allocate *variables* and to assign the *values* for them so that these values can be analysed and used later. The variables have different types: real, integer, character (byte) and string. The usage of the variables and the values assigned to the variables are to be in agreement with their type.

The operator which allocates a new variable has a form:

`(VARIABLE (`*var-type var-name var-value*`))`

where

- *var-type* is one of the identifiers `REAL`, `INTEGER`, `CHARACTER` (`CHAR`, `BYTE`) or `STRING`, which define the type of the variable;
- *var-name* is the name of the variable which is to be different from the names of the other variables;
- *var-value* is the real, 32-bit integer, 8-bit integer or string expression which defines the initial value of the variable.

Example:

```
(VARIABLE (CHARACTER Del D 127))
(VARIABLE (REAL Kern#KKK R -0.3333))
```

It is possible to allocate more than one variable inside one operator `VARIABLE` by specifying several sublists inside it:

```
(VARIABLE
    (CHARACTER Del D 127)
    (REAL Kern#KKK R -0.3333)
    ......
)
```

The variable names are case-sensitive: the variables with the names `Var` and `vAr` are different. The variable names can be composed using *arbitrary* characters, not only latin letters and digits. Nevertheless, it is better not to use the names started from '`@`' — otherwise it is possible to get a strange behaviour for the prefix commands `V@` and `@V` (the arguments of these commands are analyzed as string expressions, and in this case the first letter '`@`' is recognized as the prefix of string operastion).

If the operator `VARIABLE` uses the name which was already used, the warning message is printed and the old variable is deleted. The variable value can be changed without the warning message by the operator `ASSIGN` which has the form

`(ASSIGN` *var-name var-value*`)`

The type of the value *var-value* is to be in agreement with the trype of the variable defined earlier. Only one variable can be changed in one operator `ASSIGN`. Example:

```
(ASSIGN Del D 255)
(ASSIGN Kern#KKK R +1.51134)
```

It is possible also to allocate new vartiables and to assign their initial values directly from the command line — see section 1.2 for more details.

2.1.4 Conditional operators

VFComb script files can contain the conditional `IF-THEN-ELSE` operators. The following groups of conditional operators can be used:

- comparison of integer values;
- comparison of real values;
- comparison of string values;
- operators which check the existance of the variables;
- integer `IF-CASE` operators;
- string `IF-CASE` operators.

The depth of `IF-THEN-ELSE`-operators cannot exceed 50.

The general form of `IF`-operators is the following:

```
(IF-condition  if-data)
        .........
        commands
        .........
(ELSE)
        .........
        commands
        .........
(ENDIF)
```

The general form of `IF-CASE`-operators is the following:

```
(IF-CASE-condition  if-case-data)
        (CASE  case-value1)
        .........
        commands
        .........
        (BREAK)
        (CASE  case-value2)
        .........
        commands
        .........
        (BREAK)
        .........
(ELSE)
        .........
        commands
        .........
(ENDIF)
```

As usually, the `ELSE`-block can be skipped together with the command `(ELSE)`.

Actually `IF`-commands and `IF-CASE`-commands are the preprocessor operators which are used by *VFComb* read-data subroutines to skip some blocks of input data. `IF`-commands transfer the block after `IF`-command if some condition is fulfilled, and transfer `ELSE`-block if it is not fulfilled. `IF-CASE`-operators check that *if-case-data* is equal to *case-value* and transfer the corresponding block of commands further. The `CASE`-block is to be terminated by `(BREAK)` — like *C* conditional operators the `CASE`-block is not terminated automatically when the next `(CASE ...)` is encountered.

The following `IF`-conditions and `IF-CASE`-conditions can be used:

`(IF` *integer-value*`)` — checks that some integer expression is not equal to zero;

`(IF-NOT` *integer-value*`)` — checks that some integer expression is equal to zero;

`(IF-ONE` *integer-value*`)`, `(IF-TRUE` *integer-value*`)` — just the same as the command `(IF` *integer-value*`)`;

`(IF-ZERO` *integer-value*`)`, `(IF-FALSE` *integer-value*`)` — just the same as the command `(IF-NOT` *integer-value*`)`;

`(IF-DEF` *var-name*`)` — checks that some variable is defined already;

`(IF-NDEF` *var-name*`)` — checks that some variable is not defined still;

`(IF-EQ` $integer_1$ $integer_2$`)` — checks that $integer_1 = integer_2$;

`(IF-NE` $integer_1$ $integer_2$`)` — checks that $integer_1 \neq integer_2$;

`(IF-GT` $integer_1$ $integer_2$`)` — checks that $integer_1 > integer_2$;

`(IF-LT` $integer_1$ $integer_2$`)` — checks that $integer_1 < integer_2$;

`(IF-GE` $integer_1$ $integer_2$`)` — checks that $integer_1 \geq integer_2$;

`(IF-LE` $integer_1$ $integer_2$`)` — checks that $integer_1 \leq integer_2$;

`(IFR-EQ` $real_1$ $real_2$`)` — compare two real values: $real_1 = real_2$;

`(IFR-NE` $real_1$ $real_2$`)` — compare two real values: $real_1 \neq real_2$;

`(IFR-GT` $real_1$ $real_2$`)` — compare two real values: $real_1 > real_2$;

`(IFR-LT` $real_1$ $real_2$`)` — compare two real values: $real_1 < real_2$;

`(IFR-GE` $real_1$ $real_2$`)` — compare two real values: $real_1 \geq real_2$;

`(IFR-LE` $real_1$ $real_2$`)` — compare two real values: $real_1 \leq real_2$;

`(IFS-EQ` $string_1$ $string_2$`)` — compare two text strings: $string_1 = string_2$;

`(IFS-NE` $string_1$ $string_2$`)` — compare two text strings: $string_1 \neq string_2$;

`(IFS-GT` $string_1$ $string_2$`)` — compare two text strings: $string_1 > string_2$;

`(IFS-LT` $string_1$ $string_2$`)` — compare two text strings: $string_1 < string_2$;

`(IFS-GE` $string_1$ $string_2$`)` — compare two text strings: $string_1 \geq string_2$;

`(IFS-LE` $string_1$ $string_2$`)` — compare two text strings: $string_1 \leq string_2$;

`(IF-CASE` *integer-value*`)` — integer `IF-CASE`-operator that checks that the *integer-value* is equal to the argument of some `(CASE` *integer*`)` command in its body;

`(IF-CASE` *integer-value*`)` — integer `IF-CASE`-operator that checks that the *integer-value* is equal to the argument of some `(CASE` *int-value*`)` command specified after it and before closing `(ENDIF)` operator;

`(IFS-CASE` *string-value*`)` — string `IF-CASE`-operator that checks that the *string-value* is equal to the argument of some `(CASE` *string*`)` command specified after it and before closing `(ENDIF)` operator.

Similarly to prefix commands `V@` and `@V` (see sections 2.1.1 and 2.1.2), the arguments of the conditions `IF-DEF` and `IF-NDEF` are analyzed as the string expressions, not as the string constants (see section 2.1.2). The arguments of the string `IFS`-commands and `IFS-CASE`-command are also processed as the string expressions and the strings are converted to uppercase form before comparison.

Examples:

```
(IF-DEF FONTNAME)
      (MESSAGE The variable FONTNAME = @V FONT)
(ELSE)
      (VARIABLE (STRING FONTNAME CMR10))
(ENDIF)

(IF-TRUE V monospace)
      (MESSAGE 'MONOSPACE' mode is active)
(ELSE)       (LOADFILE kerndata.tbf)
(ENDIF)

(IF-CASE V FONTPITCH)
(CASE D 5)
      (ASSIGN u# R 12.5
```

```
      (BREAK)
(CASE D 6)
      (ASSIGN u# R 14.0
      (BREAK)
.........
(ELSE)
      (MESSAGE Unknown FONTPITCH value)
      (HALTPGM)
(ENDIF)
```

2.1.5 Other operators

There are other commands which can be useful for the logical structuring of the input stream of *VFComb* commands:

`(COMMENT` *text*`)` — skip the *text* which can include arbitrary *VFComb* commands provided that the opening and closing braces '`(`' and '`)`' are balanced correctly.

`(MESSAGE` *text*`)` — print the *text* which may be the combination of string elements and string expressions (see section 2.1.2) separated by spaces, and can include opening and closing braces '`(`' and '`)`' if they are balanced correctly, but it cannot exceed 255 characters.

`(TYPEOUT` *text*`)` — just the same as `(MESSAGE` *text*`)`.

`(LOADFILE` *filename*`)` **or** `(LOAD` *filename*`)` — insert at this place the contents of the file *filemane* (*filename* can be a string expression) from the beginning and up to the end or up to the first command `(ENDLOAD)` encountered in it. If the extension of the *filename* is not specified explicitly, the extension `.tbf` is added if the command `LOADFILE` is encountered during the processing of script file `.tbf`, and the extension `.pl` is added if the command `LOADFILE` is encountered during the processing of the `.pl` corresponding to a real font metric data.

`(ENDLOAD)` — stop to read the contents of the current input file and return the the previous file.

`(ENDINPUT)` — stop to read the input data stream;

`(HALTPGM)` — stop the program.

2.2 `MAPFONT` or `FONT`

The command `MAPFONT` describes the real fonts which are used in the virtual font. One `MAPFONT` command should be specified for each real font used in the virtual font.

The command `MAPFONT` assignes the integer *index* to the real font, and all the commands which use the references to the real fonts refer to this *index* value. It can also specify the processing modes for this font (some of them overwrites the modes specified at the command `OPTION` described at the section 2.7.6) and the standard mapping table for the characters extracted from this font — so that the commands `CHARACTER` deascribed at section 2.3 becomes unnecessary to great extend.

The command `MAPFONT` has the format:

`(MAPFONT` *index* `(`*option*$_1$`) (`*option*$_2$`)...  )`

where the *index* is the integer value. The valid *options* are:

`(FONTNAME` *filename*`)` **or** `(NAME` *filename*`)` — specifies the name of the real font (obligatory parameter).

`(TFMINPUT` *filename*`)` — specifies the name of the `.pl` file which is used to load data about this font. The `.pl` file <`fontname`>`.tbf` is used if this parameter is not specified where <`fontname`> is value of the parameter `FONTNAME` defined above.

The command `(TFMINPUT EMPTY)` means that this is the font which contains no characters at all. The command `(TFMINPUT DUMMY)` means that this is the font which contains all the characters but with zero metric information and without `VARCHAR` and `NEXTLARGER` attributes. In both cases no `.pl` file is read by *VFComb* — the necessary font information is generated automatically.

`(FONTAT` *real-value*`)` — specifies the parameter `FONTAT` transferred to `.vpl` file (the default value `(FONTAT R 1.0)`). This parameter is also used to re-calculate *dimensional* parameters from the `.pl` file according to `DESIGNUNITS` and `DESIGNSIZE` from the header of the `.vpl` file.

`(FONTAREA` *text-string*`)` — specifies the parameter `FONTAREA` transferred to `.vpl` file. This parameter specifies the explicit directory for corresponding real font, and as a rule is not used since it restricts the flexibility of font specification [1].

`(CHECKSUM` *32-bit-integer*`)` — specifies explicitly the value `FONTCHECKSUM` — the *checksum* of the `.tfm` file of this real font stored in the `.vpl` file. Usually it is not necessary to specify this parameter explicitly since its correct value is copied from the `.pl` file with the font metric data. When the *DVI*-driver loads the virtual and the real fonts, the `FONTCHECKSUM` from the `.vpl` file is compared with the *checksum* calculated for the `.tfm` files, and the warning message is printed if these values are different (this message shows that the versions of `.vf` and `.tfm` files may be different or that the `.tfm` file may be damaged).

`(FULL)` , `(7TO8)`, `(8TO7)`, `(7TO7)`, `(8TO8)`, `(JWNCYR)`, `(CYRTUG)` — these parameters specify the default mapping table for this real font: according to some rule the characters of the virtual font are mapped into the characters of this real font if these characters of the real font are defined and if the corresponding positions of the virtual font are not occupied still.

- `(FULL)` — include in the virtual font all the characters from this font so that the character codes are mapped to themselves;
- `(7TO8)` — include in the virtual font the characters 0–127 from the real font so that the character `D 0` of the real font corresponds to the character `D 128` of the virtual font, the character `D 1` of the real font corresponds to the character `D 129` of the virtual font, etc.;
- `(8TO7)` — include in the virtual font the characters 128–255 from the real font so that the character `D 128` of the real font corresponds to the character `D 0` of the virtual font, the character `D 129` of the real font corresponds to the character `D 1` of the virtual font, etc.;
- `(7TO7)` — include in the virtual font the characters 0–127 from this font so that the character codes are mapped to themselves;
- `(8TO8)` — include in the virtual font the characters 128–255 from this font so that the character codes are mapped to themselves;
- `(CYRTUG)` — include in the virtual font the characters 128–175, 224–244 from this font so that the character codes are mapped to themselves (*CYRTUG* coding scheme for cyrillic characters);
- `(JWNCYR)` — include in the virtual font the characters 128–175, 224–244 from this font so that the cyrillic characters for *WNCYR* coding scheme are mapped to cyrillic characters for *CYRTUG* coding scheme.

`(TRACE)`, `(NOTRACE)`, `(TRACE D` *integer*`)` — specifies the level for printing of the debugging information when the ligature tables of this font are processed by *VFComb*.

`(LIGTABLE)` **and** `(NOLIGTABLE)` — include or do not include in the virtual font the ligature tables of this font.

`(PHANTOM)` — assign attribute `PHANTOM` to all characters which are mapped to this font, for which the attributes `PHANTOM` or `NOPHANTOM` are not specified explicitly.

`(NOPHANTOM)` — switches off for this font the option `PHANTOM` specified by the command `OPTION` (see section 2.7.6).

`(AUTOTFMWIDTH)` **and** `(NOAUTOTFMWIDTH)` — for the characters which are mapped to this font and for which the attributes `SIZECHAR` and `SIZEFONT`

are specified, *VFComb* adds automatically the attribute `AUTOTFMWIDTH` or the attribute `NOAUTOTFMWIDTH`, respectively, if such attribute is not specified explicitly (see command `CHARACTER` (section 2.3 for more details).

`(AUTOTFMCHAR` *font-index*`)` — for the characters which are mapped to this font without explicit attributes `TFMFONT`, `TFMCHAR` or `NOTFMDATA` (see command `CHARACTER` (section 2.3 for more details), the attributes

`(TFMFONT D` *font-index*`) (TFMCHAR D` *char-code*`)`

are added automatically.

`(AUTOTFMMAPCHAR` *font-index*`)` — for the characters which are mapped to this font without explicit attributes `TFMFONT`, `TFMCHAR` or `NOTFMDATA` (see command `CHARACTER` (section 2.3 for more details), the attributes

`(TFMFONT D` *font-index*`) (TFMCHAR D` *mapchar-code*`)`

are added automatically.

`(INCLUDECHAR (`$ident_1$`)(`$ident_2$`) ...)` — specifies for this font the mode of automatical including new charascters. The following *ident* values can be used:

- `ALLCHAR` — include automatically all the characters from the real fonts even it is not specified explicitly in `CHARACTER` or `MAPFONT` commands;
- `LIGTABLE` — include in the virtual font the characters which are listed in the ligature tables specified for already included characters;
- `KRNDATA` — include in the virtual font the characters which are listed in a kerning data of the ligature tables specified for already included characters;
- `LIGDATA` — include in the virtual font the characters which are listed in a ligature data of the ligature tables specified for already included characters;
- `NOLIGTABLE` — do not include in the virtual font automatically the characters from the ligature table;
- `NEXTCHAR` — include automatically new characters which are are connected by the attribute `NEXTLARGER` with the already included characters;
- `NONEXTCHAR` — do not include new characters from the real fonts which are connected by the attribute `NEXTLARGER` with the already included characters.

`(NEWDISCARD)` — the underfined characters of this font are marked automatically by the attribute `DISCRAD` in the global mapping table (see commands `CHARACTER` (section 2.3) and `DISCARDCHAR` (section 2.7.5)).

`(NONEWDISCARD)` — the underfined characters of this font are *not* marked automatically by the attribute `DISCRAD` in the global mapping table (see commands `CHARACTER` (section 2.3) and `DISCARDCHAR` (section 2.7.5)).

Examples:

```
(MAPFONT D 0 (NAME CMR10) (7TO7) (NOLIGTABLE) ...  )
(MAPFONT D 1 (NAME LHR10) (JWRCYR) (INCLUDELIG) ...  )
......
```

2.3 CHARACTER or CHAR

The command `CHARACTER` specifies the mapping table between the characters of the real fonts and the characters of the virtual fonts. It can also specify

- the sequence of *DVI*-commands substituted instead of the character from the virtual font;
- the "phantom" output for some characters of the virtual font;
- the dimensional parameters (width, height, depth, italic correction) of the character if they are not copied from the real font metric data and are not calculated automatically for the sequences of *DVI*-commands;
- the correction of the *DVI*-pen position if the width of the character from the real font is not equal to the width specified in `.tfm` file of the virtual font;
- special attributes which are treated by *VFComb* then the default mapping tables (see command `MAPFONT` from the section 2.2) and the automatical including of new characters joined with already included characters (see the commands `MAPFONT/INCLUDECHAR` (section 2.2) and `OPTION/INCLUDECHAR` (section 2.7.6) are used.

The command `CHARACTER` has the following format:

`(CHARACTER` *char-code* `(`$parm_1$`) (`$parm_2$`) (`$parm_3$`) ...)`

The 8-bit integer value *char-code* (see section 2.1.1) defines the character of the virtual font which properties are defined by the sub-list *parameters*. The *parameters* can be distributed among several `CHARACTER` commands with the same 8-bit code — in this case they are joined together in one list.

The following *parameters* can be used:

`(FONT` *index*`)` **or** `(SELECTFONT` *index*`)` — specifies the real font into which this character is mapped. The real font is identified by the integer value *index*, and the relation between the *index* and the name of the real font is defined by the command `MAPFONT` (see section 2.2). The default value for `FONT` is set by the command `SETMAPFONT` (see section 2.7.4).

(CHAR *char-code*) **or** (SETCHAR *char-code*) — specifies the character of the real font into which this character is mapped (the real font is defined by the command FONT described above). If the metric data for this character are not specified explicitly in .tbf file (see the commands below), it is copied from the .pl file corresponding to the real font — or from the character from the real font which is specified by the commands TFMFONT and TFMCHAR if these attributes are defined.

(CHARWD *real-value*) — specifies the *width* of the virtual character.

(CHARHT *real-value*) — specifies the *height* of the virtual character.

(CHARDP *real-value*) — specifies the *depth* of the virtual character.

(CHARIC *real-value*) — specifies the *italic correction* of the virtual character.

(TFMFONT *index*) **and** (TFMCHAR *char-code*) — these two commands specify the real font and the character from which the metric information is copied. Their syntaxis is similar to the commands FONT/CHAR described above. The default value for TFMFONT is set by the command SETTFMFONT (see section 2.7.4).

The attributes TFMFONT/TFMCHAR are useful if it is necessary to make the virtual fonts which are used for visualization of the .dvi file with PostScript fonts which are substituted by some .pk fonts with a different metric properties (see section 3.3 for more details). The commands MAPFONT (section 2.2) and OPTION (section 2.7.6) can specify these atributes by default for the character if there are no explicit commands TFMFONT, TFMCHAR or NOTFMDATA.

(NOTFMDATA) — specifies that the attributes TFMFONT and TFMCHAR cannot be added automatically to this character (see corresponding arguments of the commands MAPFONT (section 2.2) and OPTION (section 2.7.6)).

(AUTOTFMWIDTH) **and** (NOUTOTFMWIDTH) — these commands switch on/off the automatic correction of the width of the virtual character if the metric information from .vpl and .tfm files for the virtual font is different from that specified for the real font characters (see the commands TFMFONT/TFMCHAR described above and the section 3.3 for more information).

The default value of this attribute can be specified using the commands MAPFONT (section 2.2) and OPTION (section 2.7.6). If no (AUTOTFMWIDTH) and (NOAUTOTFMWIDTH) attributes are specified using these commands, the attribute (AUTOTFMWIDTH) is assumed.

(PKWIDTH *width*) — specifies the width of the character (*width* is the positive real value) which is used to correct the *DVI*-pen position when the

attribute AUTOTFMWIDTH is active and the commands TFMFONT/TFMCHAR are specified. The value specified by PKWIDTH substitutes the data taken from the character referenced by the commands FONT/CHAR or the data calculated by *VFComb* for the sequence of *DVI*-commands (see the command SETDVI described below).

(DVI *list-of-arguments*) **or** (SETDVI *list-of-arguments*) — specifies the sequence of *DVI*-commands substituted instead of the character of the virtual font. The arguments of this command are described in section 2.3.1.

The metric data for this virtual character can be specified explicitly by the commands CHARWD, CHARHT, CHARDP, CHARIC described above. if these data are absent,

- if the commands FONT and CHAR or if the commands TFMFONT and TFMCHAR are defined, the metric data for the virtual character is copied from corresponding character of the corresponding .pl file.
- if commands FONT/CHAR and TFMFONT/TFMCHAR are absent, the values CHARWD, CHARHT and CHARDP are calculated automatically according to the sequence of *DVI*-commands, and CHARIC is set to zero.

If the commands TFMFONT/TFMCHAR are specified and if AUTOTFMWIDTH mode is active (by default or explicitly specified), the correction of the width of the virtual character is is performed using

- the value of the parameter PKWIDTH if it is specified explicitly;
- the width of the character specified by the commands FONT/CHAR if these commands are defined for the virtual character;
- the width of the the sequence of *DVI*-commands which is calculated automatically by *VFComb*.

(VARCHAR *list-of-arguments*) — specifies the extensible character (usually the mathematical delimiter). The arguments of this command are described in section 2.3.2. The command VARCHAR cannot be used together with the command NEXTLARGER.

(NEXTLARGER *char-code*) — specifies the character (usually the mathematical delimiter) which is "larger" than the current character. The references NEXTLARGER cannot form the infinite cycles. The command NEXTLARGER cannot be used together with the command VARCHAR.

(DISCARD) — specify the attribute DISCARD for the virtual font character which means that this character cannot be used by default mapping tables (see command MAPFONT) and by automatic including of new virtual font characters (see commands MAPFONT/INCLUDECHAR (section 2.2) and OPTION/INCLUDECHAR (section 2.7.6).

`(UNKNOWN)` — default attribute which means that the character is not used by the virtual font, but it can be added automatically when the default mapping table is used (see command `MAPFONT` described at the section 2.7.6) or during the automatic including of new characters to the virtual font.

`(PHANTOM)` — specifies that the virtual character is displayed as a blank space. It means that instead of the output of the real font character or of the sequence of *DVI*-commands, the shift of the *DVI*-pen position at a distance equal to the width of the virtual character is performed. Corresponding *DVI*-commands are generated and inserted in `.vpl` file automatically by *VFComb*. The attribute `PHANTOM` can be specified by defaul using the commands `MAPFONT` (see section 2.2) and `OPTION` (see section 2.7.6).

`(NOPHANTOM)` — swithches off the attribute `PHANTOM` specified by default with the help of the commands `MAPFONT/PHANTOM` and `OPTION/PHANTOM`.

Similar to dimensional data specified for the `.tbf` commands `LIGTABLE`, `HEADER`, etc., the dimensional parameters of the commands `CHARWD`, `CHARHT`, `CHARDP`, `CHARIC`, `PKWIDTH` and of the sub-commands of the command `SETDVI` are specified using the values `TBFUNITS` and `TBFSIZE` from the `.tbf` file (see section 2.7.2).

2.3.1 `CHARACTER/SETDVI`

The parameter `DVI` of the command `CHARACTER` specifies the the sequence of *DVI*-commands substituted instead of the character from the virtual font. It has the format

```
(SETDVI
      (command1)
      (command2)
      ......
)
```

where the following *commands* can be used:

`(FONT` *index*`)` **or** `(SELECTFONT` *index*`)` — makes this font active for the subsequent `CHAR` commands. The font is identified by the integer value *index*, and the name of the font with this index is defined using the command `MAPFONT` (see section 2.2). if there are commands `CHAR` in the sequence of *DVI*-commands, the command `FONT` is to be used before the first `CHAR`.

`(CHAR` *char-code*`)` **or** `(SETCHAR` *char-code*`)` — output the character *char-code* from the real font specified by the last command `FONT` starting from the current *DVI*-pen position and to move the *DVI*-pen position to the right at the distance equal to the width of the character.

`(RULE` *height width*`)` **or** `(SETRULE` *height width*`)` — output the *rule* (i.e., a black rectangle) with the specified height and width (*height* and *width* are positive real values) starting from the current *DVI*-pen position and to move the *DVI*-pen position to the right at the distance equal to the rule width.

`(RIGHT` *dist*`)` **or** `(MOVERIGHT` *dist*`)` — move current *DVI*-pen position to the *right* at the specified distance *dist* which is to be a real (positive or negative) value.

`(LEFT` *dist*`)` **or** `(MOVELEFT` *dist*`)` — move current *DVI*-pen position to the *left* at the specified distance *dist* which is to be a real (positive or negative) value.

`(UP` *dist*`)` **or** `(MOVEUP` *dist*`)` — move current *DVI*-pen position *up* at the specified distance *dist* which is to be a real (positive or negative) value.

`(DOWN` *dist*`)` **or** `(MOVEDOWN` *dist*`)` — move current *DVI*-pen position *down* at the specified distance *dist* which is to be a real (positive or negative) value.

`(PUSH)` — remember the current *DVI*-pen position (to be restored by the subsequent command `(POP)`.

`(POP)` — restore the *DVI*-pen position saved earlier by the command `(PUSH)`. The commands `(PUSH)` and `(POP)` should be properly nested and balanced like the parentheses.

`(SPECIAL` *text-string*`)` — the *text-string* is interpreted similar to the command `\special{...}` from `.dvi` file.

`(HEX` *hex-text-string*`)` **or** `(SPECIALHEX` *hex-text-string*`)` — the sequence of bytes identified by hexadecimal codes is treated as the argument of the command `\special{...}` from `.dvi` file. The *hex-text-string* cannot contain spaces, it should contain only hexadecimal digits '`0`'–'`9`' and '`A`'–'`F`' or '`a`'–'`f`' and it should contain the even number of such digits.

Similar to dimensional data specified for the commands `CHARACTER`, `LIGTABLE`, `HEADER`, the dimensional parameters of the commands `RULE`, `RIGHT`, `LEFT`, `UP`, `DOWN` are specified using the values `TBFUNITS` and `TBFSIZE` from the `.tbf` file (see section 2.7.2).

2.3.2 `CHARACTER/VARCHAR`

The parameter `VARCHAR` of the command `CHARACTER` specifies the extensible character (usually the mathematical delimiter). Its parameters define the character codes which are used as the top of the delimiter, middle of the delimiter, bottom of the delimiter and the replicated part of the delimiter. The zero value

of corresponding character codes (except replicated part) means that this part of the extensible character is absent.

The command has the format

`(VARCHAR (`*parm*$_1$`) (`*parm*$_2$`) ...)`

where the following parameters can be used:

`(TOP` *char-code*`)` — specifies the character which is used as the top element of the extensible character;

`(MID` *char-code*`)` — specifies the character which is used as the middle element of the extensible character;

`(BOT` *char-code*`)` — specifies the character which is used as the bottom element of the extensible character;

`(REP` *char-code*`)` — specifies the character which is used as the replicated part of the extensible character.

2.4 LIGTABLE

The command `LIGTABLE` specifies the additional ligature and kerning data for the virtual font characters — mainly the ligature and kerning data for the pairs of characters selected from different real fonts. Notot too much `LIGTABLE` commands are necessary in the `.tbf` file since the ligature data for the pairs of characters selected from the same real font are usually copied from corresponding `.pl` files.

The command `LIGTABLE` has the format:

```
(LIGTABLE
        (ligature-command1)
        (ligature-command2)
        ......
)
```

It several `LIGTABLE` commands are specified, they are joined together after reading the `.tbf` file.

The *ligature-commands* are similar to that specified at `.pl` files except the fact that the *dimensional* parameter of the command `KRN` is specified using the values `TBFUNITS` and `TBFSIZE` (see section 2.7.2). The following commands are valid here:

`(LABEL` *char-code*`)` — start the list of ligature commands for the character *char-code* of the virtual font;

`(LABEL BOUNDARYCHAR)` **or** `(LABEL BCHAR)` — start the list of ligature commands for the special character "*end-of-word*";

(SKIP *integer*) — jump over specified LIGTABLE commands LIG and KRN;

(STOP) — stop the *LIGTABLE* commands started by the command *(LABEL* char-code*)*;

(KRN *char-code′ shift*) — the kerning data *shift* (real value) specified for the pair of characters *char-code* and *char-code′* (the value *char-code* is specified at the command LABEL which starts the sequence of ligature and kerning commands for this character);

(LIG *char-code′ char-code″*) — the ligature data for the pair of characters *char-code* and *char-code′* (the value *char-code* is specified at the command LABEL which starts the sequence of ligature and kerning commands for this character) — these two characters are substituted by the character *char-code″*. The identifiers

/LIG, /LIG>, LIG/, LIG/>, /LIG/, /LIG/>, /LIG/>>

can be used instead of LIG which defines various ways of the substitution of the character *char-code″* instead of the pair of characters *char-code* and *char-code′* (see [1] for more details).

Example:

```
(LIGTABLE
(LABEL C <)
      (LIG C < V CYR_open_quote)
      (STOP)
(LABEL C >)
      (LIG C > V CYR_close_quote)
      (STOP)
(LABEL V CYR_GHE)
      (KRN C . V kk#)
      (KRN C , V kk#)
      (KRN C : V kk#)
      (KRN C ; V kk#)
      (STOP)
)
```

2.5 OUTPUT

The command OUTPUT defines the name of the output .vpl file and, may be, the way how the character codes for the .vpl commands are to be printed. It has the form

(OUTPUT *filename* (*option*$_1$) (*option*$_2$) ...)

where *filename* is the name of the output *virtual font*, and the optional *options* can be the following:

`(ALLCHAR)` — print all printable characters in a form `C` *letter*;

`(ALPHA)` — print in a form `C` *letter* only the letters of the latin alphabet;

`(ALPHANUM)` — print in a form `C` *letter* the digits and the letters of the latin alphabet;

`(ALLCODE)` — print all characters as the character codes;

`(DEC)` **or** `(DECIMAL)` — print character codes as decimal integers;

`(OCT)` **or** `(OCTAL)` — print character codes as octal integers;

`(HEX)` **or** `(HEXADECIMAL)` — print character codes as hexadecimal integers.

Example (default value):

```
(OUTPUT noname.tbf (ALPHANUM) (OCTAL))
```

2.6 HEADER

The virtual font has the header with the header parameters:

- `DESIGNSIZE`
- `DESIGNUNITS`
- `CODINGSCHEME`
- `FAMILY`
- `FACE`
- `FONTDIMEN` and its sub-parameters (`SLANT`, `SPACE`, `QUAD`, ..)

The header parameters can be get from:

- the header of one of the real fonts used in virtual font;
- the header of some separate `.pl` file;
- *VFComb* the explicit commands from the `.tbf` file.

The following parameters of the command `HEADER` are valid:

(FONT *integer*) — defines index of the font which is used as a source for the header parameters. The real name of the font is taken from the MAPFONT command with the corresponding index value. Default value for this parameter is (FONT D 0) which means that if nor FONT nor FILE dat are specified by the User, the font MAPFONT D 0 is used as a source for the header parameters. If FONT argument is <0 or if the parameter FILE is specified, the MAPFONT fonts are not used as a source for the header.

(FILE *filename*) — defines the name of the .pl file which is used as a source for the header parameters. Only one of the parameters — FONT or FILE — can be specified among the parameters of the command HEADER. If both parameters are specified, the last one is used and the warning message is generated.

(CHECKSUM *32-bit integer*) — specifies the CHECKSUM of the .tfm file which will be generated after the processing of .vpl file by the utility *VPtpVF*. Usually this parameter is not necessary — if it is not specified, the correct CHECKSUM value is substituted by *VPtoVF* automatically, and it is a rare chance that the User can guess the proper value (see comments below).

(DESIGNSIZE *real-value*) — specifies the value of the DESIGNSIZE which is used as a unit of measure for nearly all dimensional .vpl data (see comments below).

(DESIGNUNITS *real-value*) — specifies the value of the DESIGNUNITS which is used as a unit of measure for nearly all dimensional .vpl data (see comments below).

(CODINGSCHEME *text-string*) — specifies the text which identifies uniquely the arrangement of the characters in the coding table of this font. For example, for Computer Modern Roman except cmr5 it is equal to

```
(CODINGSCHEME TeX text)
```

while for cmr5 it is equal to

```
(CODINGSCHEME TeX text without f-ligatures)
```

(the *f-ligatures* are absent in this font since the characters are too small). This parameter cannot contain internal braces '(' and ')' and but can be the string expression (see section 2.1.2 for more details).

(FAMILY *text-string*) — specifies the FAMILY identifying string. This parameter cannot contain internal braces '(' and ')' but can be the string expression (see section 2.1.2 for more details).

(FACE *8-bit-integer*) — specifies the Xerox face code for the virtual font.

(SEVENBITSAFEFLAG) — this command is of no use and its is skipped after printing the warning message. The attribute SEVENBITSAFEFLAG was used in earlier versions of TEX to specify that the font contains only the characters in a range 0–127, and now this flag is obsolete. Although the corresponding field is conserved in .tfm file, the utility *VPtoVF* inserts the proper value automatically (after the check that the virtual font contains no characters outside the range 0–127) irregardless of the command SEVENBITSAFEFLAG encountered in .vpl file.

(FONTDIMEN *parm-list*) — defines the following header parameters:

(SLANT *real-value*)
(SPACE *real-value*)
(STRETCH *real-value*)
(SHRINK *real-value*)
(XHEIGHT *real-value*)
(QUAD *real-value*)
(EXTRASPACE *real-value*)
(NUM1 *real-value*) or (DEFAULTRULETHICKNESS *real-value*)
(NUM2 *real-value*) or (BIGOPSPACING1 *real-value*)
(NUM3 *real-value*) or (BIGOPSPACING2 *real-value*)
(DENOM1 *real-value*) or (BIGOPSPACING3 *real-value*)
(DENOM2 *real-value*) or (BIGOPSPACING4 *real-value*)
(SUP1 *real-value*) or (BIGOPSPACING5 *real-value*)
(SUP2 *real-value*)
(SUP3 *real-value*)
(SUB1 *real-value*)
(SUB2 *real-value*)
(SUPDROP *real-value*)
(SUBDROP *real-value*)
(DELIM1 *real-value*)
(DELIM2 *real-value*)
(AXISHEIGHT *real-value*)
(AXISHEIGHT *real-value*)

(the parameters specified in a form (PARAMETER *index value*) are skipped and the warning message is printed).

All these parameters are *dimensional* parameter except the parameter SLANT and they are defined using TBFUNITS and TBFSIZE data from the .tbf file (see section 2.7.2). They are re-calculated properly so that they fit to the header parameters DESIGNUNITS and DESIGNSIZE when the header data of the output .vpl file are known.

The parameters are specified as the sub-lists of the list defined by the identifier HEADER. Example:

```
(HEADER
      (FONT D 1)
      (FAMILY LHTT)
      (CODINGSCHEME LH Cyrillic TeX text)
      ......
)
```

VFComb generates the header parameters in a following way. The commands from the `.tbf` file are read, and the dimensional parameters from the command `HEADER` are treated using the units defined by `TBFSIZE` and `TBFUNITS` (see section 2.7.2) — as well as the other dimensional parameters like that from the commands `LIGTABLE` and `CHARACTER`. After it the `.pl` file specified by the `HEADER` commands `FONT` or `FILE` is read, and the parameters which were not defined explicitly at the command `HEADER` are added from it. Finally, all the dimensional parameters which vere read from the `.tbf` file (namely, specified at the commands `LIGTABLE`, `CHARACTER`, `HEADER/FONTDIMEN`) are converted to `DESIGNSIZE` and `DESIGNUNITS` values from the resulting header of the `.vpl` file.

The `.pl` file header parameters

`DESIGNSIZE`
`DESIGNUNITS`
`CODINGSCHEME`
`FAMILY`
`FACE`
`FONTDIMEN` sub-parameters except `PARAMETER`

are processed by *VFComb*. The `.pl` file header parameters

`SEVENBITSAFEFLAG`
`HEADER`
sub-parameter `PARAMETER` of the parameter `FONTDIMEN`

are ignored after printing the warning messages. All other *legal* `.pl` commands are skipped without processing when *VFComb* reads the header parameters.

The header parameters which are read from `.pl` file are added to `.vpl` header data if corresponding parameters are not specified explicitly in `.tbf` file. Usually *all* header parameters (including `DESIGNUNITS` and `DESIGNSIZE`) are copied from some `.pl` file, and only the parameters `FAMILY`, `FACE`, `CODINGSCHEME` are substituted by their explicit specifications from `.tbf` file.

The exception is the header parameter `CHECKSUM` which is ignored when it is read from `.pl` file and which value can be set by the User only if it is specified explicitly in `.tbf` file. Usually this parameter is not necessary — if it is not specified by the User, it is not inserted into the output `.vpl` file, and as a result the correct `CHECKSUM` value is calculated by *VPtoVF* automatically. It is a rare

chance that the User can guess the proper value, and if the CHECKSUM from the .vf file and the *checksum* calculated for the .tfm file of the virtual font do not coincide, the *DVI*-driver prints the warning message — this message shows that the versions of .vf and .tfm files may be different or that the .tfm file may be damaged. The exception from this rule (and the only case when the explicit parameter CHECKSUM can be useful) is the *zero* value specified for the parameter CHECKSUM of the virtual font. In this case the comparison of CHECKSUM from .vf file with the the *checksum* of the .tfm file of the virtual font is not performed at all when this font is loaded by the *DVI*-driver.

2.7 Additional commands

The following *VFComb* commands are considered in this section:

- VTITLE — the title of the virtual font;
- TBFUNITS and TBFSIZE — the scale factors for the lengths specified in .tbf file;
- BOUNDARYCHAR — character code which is used in the .vpl ligature table to specify the "end-of-word" character;
- SETTFMFONT — default value for the attribute TFMFONT if for some commands CHARACTER only the attribute TFMCHAR is specified;
- SETMAPFONT — default value for the attribute FONT if for some commands CHARACTER only the attribute CHAR is specified;
- DISCARDCHAR — the list of characters which are not included in the virtual font;
- NOTAUTOADDCHAR — the list of characters which cannot be used for automatically characters;
- OPTION — modes of *VFComb*;
- SAVETABLE — printing of the mapping table into ASCII file;

2.7.1 VTITLE

The command VTITLE of the .vpl file defines the title of the file which is actually the comment inserted in the binary .vf file after processing by the utility *VPtoVF*. The command VTITLE has the form

(VTITLE *text-string*)

where the *text string* is the combination of string elements and string expressions (see section 2.1.2) separated by spaces, and can include opening and closing braces '`(`' and '`)`' if they are balanced correctly. The result of the expansion of string expressions cannot exceed 255 characters.

Example:

```
(VTITLE Created by VFComb (Version 1.3))
```

2.7.2 `TBFUNITS` and `TBFSIZE`

All the length data in `.vpl` file are defined in units specified by the parameters `DESIGNUNITS` and `DESIGNSIZE` from the `HEADER` command (see section 2.6). Nethertheless, when the User prepares the `.tbf` file, these parameters are not known yet since they can be read later from the `HEADER` file. For this reason all the length and size data in `.tbf` file are specified using internal scaling factors specified bu the commands `TBFUNITS` and `TBFSIZE` with the default values

```
(TBFSIZE R 1.0) (TBFUNITS R 1.0)
```

which are the analogs of the commands `DESIGNUNITS` and `DESIGNSIZE` from `.pl` and `.vpl` files. The length and size data specified into `.tbf` file are recalculated to the proper values `DESIGNUNITS` and `DESIGNSIZE` from the command `HEADER` when the output `.vpl` file is generated.

2.7.3 `BOUNDARYCHAR` or `BCHAR`

The LIGTABLE entries can contain ligature and kerning data with a special character "end-of-word" which defines the beginning of the word when encountered in the command `LABEL`, and the end of the word when encountered in the commands `LIG` and `KRN`. The command `BOUNDARYCHAR` defines the code assigned to this special character (usually this is the code `D 255`). As a result:

- inside `.tbf` file the `LIGTABLE` entries which contain this code are recognized as the entries with "end-of-word" character;
- inside `.pl` files the `LIGTABLE` entries with their own `BOUNDARYCHAR` codes are mapped to the code specified in `.tbf` file.

Example:

```
(BOUNDARYCHAR D 255)
(BCHAR D 255)
```

2.7.4 `SETMAPFONT` and `SETTFMFONT`

The command `SETMAPFOMT` specifies the default value for the attribute `FONT` if only the attribute `CHAR` is defined (see description of the *VFComb* command `CHARACTER` in section 2.3). It has the form

`(SETMAPFONT` *ineger*`)`

where *integer* is the index of the font listed in `MAPFONT` command which is used for `FONT` attribute.

Similarly, if the command `SETTFMFOMT` specifies the default value for the attribute `TFMFONT` if only the attribute `TFMCHAR` is defined (see description of the *VFComb* command `CHARACTER` in section 2.3). It has the form

`(SETTFMFONT` *ineger*`)`

where *integer* is the index of the font listed in `MAPFONT` command which is used for `TFMFONT` attribute.

The important difference of *VFComb* `CHARACTER` command is that if the parameter `FONT` is not specified although the parameter `CHAR` *is* specified, the value specified by the command `SETMAPFONT` is used (see section 2.7.4). The syntaxis of `.vpl` files assumes that the parameter `FONT` specified for the previous command `CHARACTER` is used while for *VFComb* commands it is not so.

Similarly, if the parameter `TFMFONT` is not specified although the parameter `TFMCHAR` *is* specified, the value specified by the command `SETTFMFONT` is used (see section 2.7.4).

Example (default values):

```
(SETMAPFONT D 0)
(SETTFMFONT D 0)
```

2.7.5 `DISCARDCHAR` and `NOTAUTOADDCHAR`

The command `DISCARDCHAR` defines the characters from the real fonts which are not included in the virtual font. This command is useful if the mapping to some font is defined by one of the options `FULL`, `7TO8`, `8TO7`, `8BIT`, `7BIT`, `JWNCYR`, `CYRTUG` (see command `MAPFONT` in section 2.2) but there are some characters which are to be excluded from the mapping list.

The command has the form:

```
(DISCARDCHAR
       (FONT font1) (CHAR char1)
       (FONT font2) (CHAR char2)
       ......
)
```

Actually one command `(FONT ...)` can be followed by several commands `(CHAR ...)` if these characters belong to the same real font. Instead of several commands `(CHAR` *charcode*`)` it is possible to use one command `(RANGE` $charcode_1$ $charcode_2$`)` — in this case all the characters from the range $charcode_1$–$charcode_2$ get the attribute `DISCARD`.

Example:

```
(DISCARDCHAR
       (FONT D 0) (CHAR D 250) (CHAR D 251) ...
       (FONT D 1) (RANGE D 128 D 255) ...
       ......
)
```

The command `NOAUTOADDCHAR` defines the characters from the virtual font which cannot be used for character mapping when *VFComb* adds automatically characters since they are connected with other characters by the attribute `NEXTLARGER` or through `LIGTABLE` data (see sections 2.2 and 2.7.6 for more details) — these characters get the attribute `DISCARD` if this character is not used explicitly in mapping table.

The command has the form:

`(NOAUTOADDCHAR (CHAR` $char_1$`) (CHAR` $char_2$`) (CHAR` $char_3$`) ...)`

Similarly to `DISCARDCHAR`, instead of several commands `(CHAR` *charcode*`)` it is possible to use one command `(RANGE` $charcode_1$ $charcode_2$`)` — in this case all the characters of the virtual font from the range $charcode_1$–$charcode_2$ get the attribute `DISCARD` if these characters are not used explicitly in mapping table.

2.7.6 `OPTION`

The command `OPTION` defines some modes of *VFComb* operations. It consists of identifiers where only `TRACE` and `INCLUDECHAR` can have a non-empty argument:

`(TRACE` *integer*`)` — specifies the level of tracing messages printed during the processing of the ligature tables (*integer* should be in a range 0–3);

`(TRACE)` — just the same as `(TRACE D 2)`;

`(NOTRACE)` — just the same as `(TRACE D 0)`;

`(SCREEN)` — print the *VFComb* messages on the display screen;

`(NOSCREEN)` — print the *VFComb* messages only into `.log` file;

`(PHANTOM)` — assign attribute `PHANTOM` to all characters, for which the attributes `PHANTOM` or `NOPHANTOM` are not specified explicitly;

`(INCLUDECHAR (`$ident_1$`)(`$ident_2$`) ...)` — specifies how *VFComb* should automatically add characters to the virtual font. The following *ident* values can be used:

`ALLCHAR` — include automatically all the characters from the real fonts even it is not specified explicitly in `CHARACTER` or `MAPFONT` commands;

`LIGTABLE` — include in the virtual font the characters which are listed in the ligature tables specified for already included characters;

KRNDATA — include in the virtual font the characters which are listed in a kerning data of the ligature tables specified for already included characters;

LIGDATA — include in the virtual font the characters which are listed in a ligature data of the ligature tables specified for already included characters;

NOLIGTABLE — do not include in the virtual font automatically the characters from the ligature table;

NEXTCHAR — include automatically new characters which are are connected by the attribute NEXTLARGER with the already included characters;

NONEXTCHAR — do not include new characters from the real fonts which are connected by the attribute NEXTLARGER with the already included characters.

(LIGTABLE) — include in the virtual font the ligature tables of the real fonts;

(NOLIGTABLE) — do not include in the virtual font the ligature tables of the real fonts;

(NEWDISCARD) — the underfined characters of the real font are marked automatically by the attribute DISCRAD in the global mapping table (see commands CHARACTER (section 2.3) and DISCARDCHAR (section 2.7.5));

(NONEWDISCARD) — the underfined characters of the real font are *not* marked automatically by the attribute DISCRAD in the global mapping table (see commands CHARACTER (section 2.3) and DISCARDCHAR (section 2.7.5));

(AUTOTFMWIDTH) — add automatically for the characters with the attributes SIZECHAR and SIZEFONT the attribute AUTOTFMWIDTH if it is not specified explicitly (see command CHARACTER);

(NOAUTOTFMWIDTH) — add automatically for the characters with the attributes SIZECHAR and SIZEFONT the attribute NOAUTOTFMWIDTH if it is not specified explicitly (see command CHARACTER);

(AUTOTFMCHAR *font-index*) — for the characters without explicitly specified attributes TFMFONT, TFMCHAR or NOTFMDATA (see command CHARACTER (section 2.3 for more details) the attributes

(TFMFONT D *font-index*) (TFMCHAR D *char-code*)

are added automatically;

(AUTOTFMMAPCHAR *font-index*) — for the characters without explicitly specified attributes TFMFONT, TFMCHAR or NOTFMDATA (see command CHARACTER (section 2.3 for more details) the attributes

(TFMFONT D *font-index*) (TFMCHAR D *mapchar-code*)

are added automatically.

Example (default values):

```
(OPTION
      (TRACE D 0)
      (SCREEN)
      (INCLUDECHAR (NOLIGTABLE))
      (INCLUDECHAR (NEXTCHAR))
      (LIGTABLE)
      (NONEWDISCARD)
      (AUTOTFMWIDTH)
)
```

2.7.7 SAVETABLE

The command SAVETABLE is used when it is desirable to have *VFComb* commands which are equivalent to the mapping table equivalent to that used to generate .vpl file. It can have the following forms:

```
(SAVETABLE filename)
(SAVETABLE filename (SHORT))
(SAVETABLE filename (FULL))
```

where the options FULL and SAVE defines the format of the output.

3 *VFComb* applications

3.1 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 syntaxis (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 files. 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 in [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 output 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* enables to perform additional operation. That is, except 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 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*, which enables to 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 to perform 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 inside the command `CHARACTER` the attributes `NEXTLARGER` and/or `VARCHAR`.

These features enable to create the desired virtual fonts for national alphabets with less efforts and with more realibility than by manual manipulations with `.pl` and `.vpl` files.

3.2 Virtual fonts for colored printing

The other problem is the application of the virtual fonts to multi-colored printing. Suppose that it is necessary to print the 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 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 pass. 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 correspond 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 *emty* character. The subdirectories for other colors are organized analogously. 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 the corresponding reconfiguration of the *DVI*-driver is performed.

If the mapping of the empty characters into the `dummy` font is performed, it results to a wrong behaviour of the *DVI*-driver: the characters in `dummy` font have zero size, and it means that the next character after the empty character is shifted to the left (as compared with the desired behaviour) on the distance equal to the width of the skipped character. To prefent this effect it is necessary to insert in the virtual font the explicit *DVI*-commands which move the current output position to the right by the distance which is the width of the skipped character. This operation is performed by *VFComb* by a single command: the User can

- assign the attribute `PHANTOM` to each *phantom* character;
- assign the attribute P̂HANTOM to the font which is used to mark colored letters;
- specify the global option `PHANTOM`;

and as a result for the necessary characters the *empty mapping* will be performed instead of mapping into the real font characters.

3.3 Substitution of `.pk` fonts instead of *Postscript* fonts

The next problem where the usage of the virtual fonts is advantageous is the visualization of the document which was compiled using PostScript fonts. Gen-

erally the scren 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 the width different from the PostScript font (the fact that they have different graphical image is not so essential). Like 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* enables to specify for the real fonts *two* PL-files with the metric information: the first one for the nominal characters which is 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 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 which correct properly the current output position are inserted in 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 it is not so, the operation of re-mapping inside already mapped font ir required which could be very complicated and result to a very complicated scheme of virtual vont generation. To solve this difficulty 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 to load this information and to correct the proper character width.

Acknowledgements

Acknowledgement I

All new improvements of *VFComb* (except English manual) are the results of the contacts and discussions which were held during the *EuroTEX-95* meeting. So I would like to thank Dr. Kees van der Laan for his giant efforts to organise the visit to the *EuroTEX-95* 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 opinion makes an impact in preparing the new version of *VFComb*. Among other persons I would like to thank Dr. Ph.Taylor and Dr. S.Znamensky for their valuable suggestions which enable to improve the program. I would like also to thank

O.A.Lapko, S.A.Strelkov and I.A.Mahovaya for their efforts spent on the Cyrillic TEX project which actually inspired our work.

Acknowledgement II

I would like to express my warmest thanks to Dr. A.Compagner (Delft University of Technology), who *did not* contributed something to this work but spent a lot of his time and efforts explaining the Phylosophy of Randomness and his approach to RNG.

Acknowledgement III

??????

Acknowledgement IV

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** (1990), No. 1, pp.13–23.

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

[3] O.Lapko. *MAKEFONT as a part of CurTUG–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.