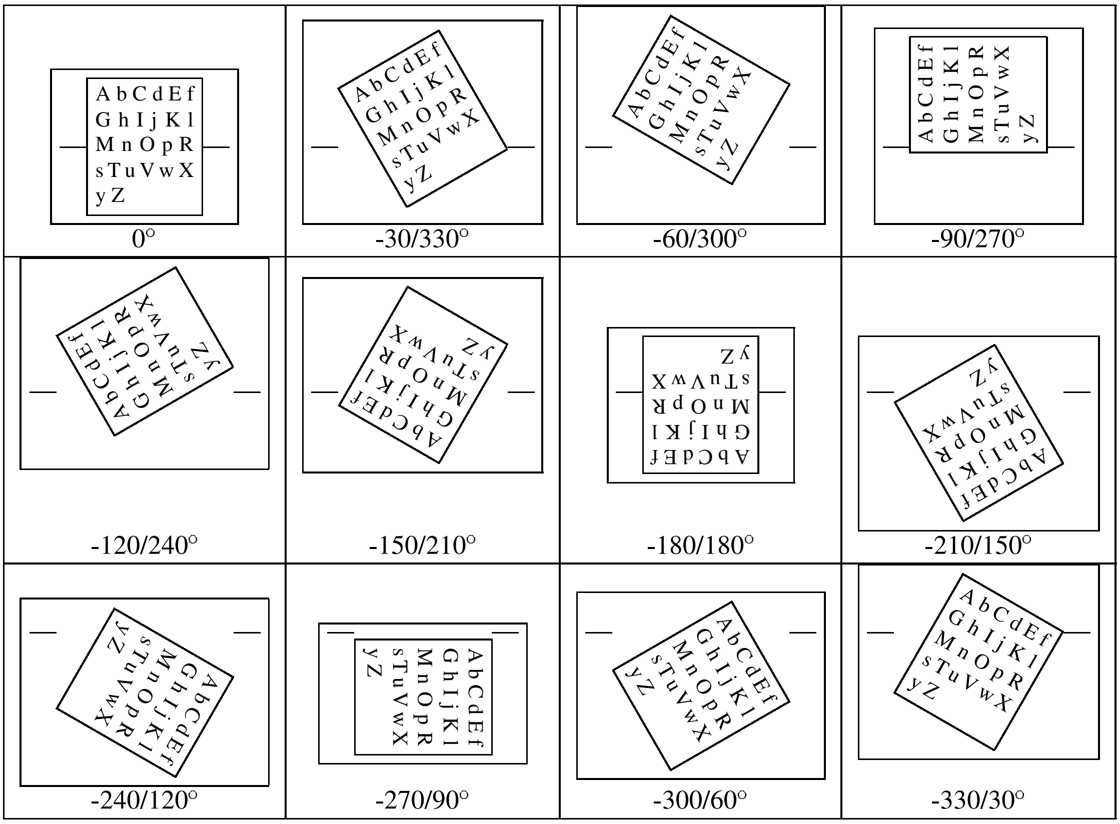
```
\begin{figure*}[p]
  \begin{minipage}[b]{.495\linewidth}
     \centering\mbox{\epsfig{figure=ps/Scandinavia.eps,width=.9\textwidth}}
     \caption{Noord Europa}   \label{fig:draai1}
  \end{minipage}\hfill
  \begin{minipage}[b]{.495\linewidth}
      \centering\mbox{\epsfig{figure=ps/Africa.eps,width=.9\textwidth}}
      \caption{De kaart van Afrika} \label{fig:draai2}
  \end{minipage}
  \centering\mbox{\epsfig{figure=ps/USA.eps,width=.70\textwidth}}
 \caption{De Verenigde Staten van Amerika}     \label{fig:draai3}
\end{figure}
```

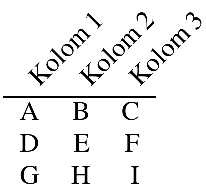
Figuur 3: *Noord Europa*

Figuur 4: *De kaart van Afrika*

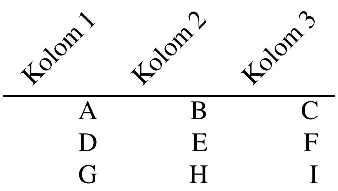
Figuur 5: *De Verenigde Staten van Amerika*



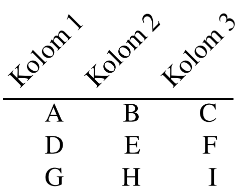
Figuur 6: *Het draaien van paragrafen*




```
\begin{tabular}{rrr}\\[3mm]
\begin{rotate}{-45}Kolom 1\end{rotate}&
\begin{rotate}{-45}Kolom 2\end{rotate}&
\begin{rotate}{-45}Kolom 3\end{rotate}\\
\hline A& B& C\\ D& E& F\\ G& H& I\\ \hline
\end{tabular}
```

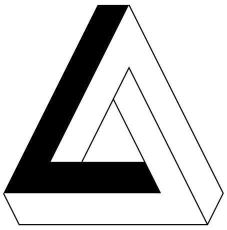



```
\begin{tabular}{rrr}
\begin{turn}{-45}Kolom 1\end{turn}&
\begin{turn}{-45}Kolom 2\end{turn}&
\begin{turn}{-45}Kolom 3\end{turn}\\
\hline A& B& C\\ D& E& F\\ G& H& I\\\hline
\end{tabular}
```




```
\begin{tabular}{rrr}\\[5mm]
\begin{rotate}{-45}Kolom 1\end{rotate}
\rule{.5cm}{0pt}&
\begin{rotate}{-45}Kolom 2\end{rotate}
\rule{.5cm}{0pt}&
\begin{rotate}{-45}Kolom 3\end{rotate}
\rule{.5cm}{0pt}\\
\hline A& B& C\\ D& E& F\\ G& H& I\\\hline
\end{tabular}
```

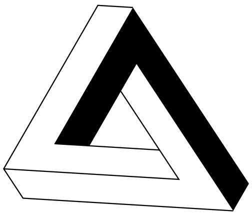
Figuur 7: *Het draaien van informatie in tabellen*



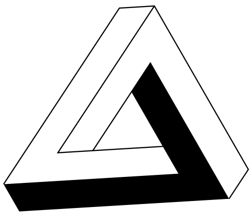
A— —B

```
A---
 \epsfig{figure=ps/Escher3.eps,width=30mm}
---B
```

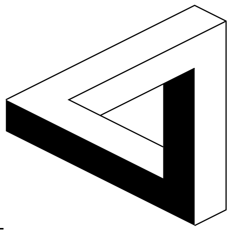
A— —B



```
A---\begin{turn}{-240}
\epsfig{figure=ps/Escher3.eps,width=30mm}
\end{turn}---B
```

A—  —B

```
A---\begin{turn}{240}
\epsfig{figure=ps/Escher3.eps,width=30mm}
\end{turn}---B
```



A— —B

```
A---\begin{sideways}
\epsfig{figure=ps/Escher3.eps,width=30mm}
\end{sideways}---B
```

Figuur 8: *Verschillende manieren om een beeld te draaien*

Virtual Fonts: Great Fun, Not for Wizards Only

Yannis Haralambous

yannis@gat.citilille.fr

1 Introduction

This paper deals with *virtual fonts*. I would like to present some examples of their astonishing possibilities, taken from everyday typesetting (or almost). First of all, what is a virtual font?

If we look at it from TEX's point of view, then we will not see any difference: for TEX, a virtual font is just like any other font. We have to admit that TEX is in a sad position: it does all this beautiful typesetting, and never sees any result, since for it a font consists just of... boxes. TEX knows only about TFM files. All the beauty of typefaces is unknown to it: the box of a Garamond 'a' looks exactly like the one for the same letter, or any other letter, in Courier[1].

So TEX will never see what it has produced; this pleasure belongs entirely to the DVI driver[2]. The driver will search for a font matching the name specified in the DVI file, pick the right character, and either display it on the screen, or print it on the page. But what kind of information is the driver looking for? Before the arrival of virtual fonts there were three main kinds of fonts: PXL fonts (these are obsolete now), PK fonts (which are still the standard "bitmap" fonts for platform-independent TEX systems) and PostScript fonts. The latter are scalable fonts, in other words, they are defined by mathematical equations just like METAFONT fonts, with the main difference that the work usually done by METAFONT, namely the rasterization of outlines, is done inside the printer, by an engine called "PostScript interpreter".

Both PK and PostScript fonts have good and bad sides. The main problem with PK fonts is that they tend to occupy a lot of space on the storage media (usually a hard disk). And the occupied space depends on the resolution of the printer: for 2540 dpi machines, PK files can get quite big. The good sides are manifold: first of all a PK file is what we get out of a METAFONT run, so usually it looks good; and METAFONT makes fonts fit to our printer: we have complete control on every pixel of our output, no matter how small this pixel can be. And of course, PK fonts can be used on every printer.

PostScript fonts cost money, at least the most commonly used ones. They work only with PostScript printers (unless we have an utility, called ATM, but even then there is no guarantee). They are send to the printer in form of curves and lines; the printer then fills these with pixels. These fonts do not have the same encoding as TEX fonts. A PostScript mechanism allows a straightforward re-encoding, usually done by the driver. Nevertheless in some situations re-encoding is not enough: if a character is not present in a font it may be available in some auxiliary font. For example, Adobe places small caps in an auxiliary font, called "Expert" font; this font does not contain uppercase letters. Making a new font, incorporating both uppercase and small cap characters can take a lot of time, and time is money. Not to mention the fact that such a font will not be a standard Adobe font and hence will have to accompany the document at all stages of production (at the risk of being lost, corrupted, substituted and so on...).

So in both cases (PK and PostScript) there is at least one big problem: either storage space, or cost.

We will show solutions to these problems, based on virtual fonts. The concept of *virtual font* is very simple: things that can appear inside a DVI file are gathered together and presented to TEX as a single character. TEX will ask for one character; the driver will replace this "virtual" character by something else: some other character (re-encoding), or a cluster of characters (for example a character plus an accent), or even a whole page (including PostScript graphic by the means of `\special` commands). We can define a font where each character is virtually mapped to a PostScript figure[3]; we can then "hyphenate" or "kern" or "perform ligatures" with figures (think of the Maya script...)

So here is already a possible construction for a virtual font: take a piece of DVI code (for example out of a DVI file) and consider this to be one character of our virtual font. This technique is used in Eberhard Mattes' QDTeXVPL. The advantage of this method: we can gather every possible TEX instruction in one character; the disadvantage: we can do no more than that. Examples: if we want a quick approximation of an 'a with acute accent', ask TEX to typeset á, by the instruction `\'a`, and borrow that code from the DVI

[1] The author axiomatically considers Adobe Times-Helvetica-Courier to be the trinity of the most annoying fonts.

[2] And to us, humans...

[3] Purists may object this example, since PostScript is by no means TEXware. The author can be excused by the fact that D. E. Knuth in his original paper on virtual fonts, gives an example containing real PostScript code.

file into our virtual font; if, for any reason, we want a 'dotless j' (for example to obtain ̂) and this character is not provided in the PostScript font, we cannot have it virtually either: there is no DVI command for erasing parts of characters[4].

The clean (but not always quick) way to produce virtual fonts, is out of VPL (Virtual Property List) files; these are PL files with some additional commands[5]

To compile PL files into TFM ones, and vice versa, we use the utilities PLtoTF and TFtoPL. The situation is similar for virtual fonts: we have tools VPtoVF and VFtoVP which convert a VPL file into a VF and a TFM file.

Let's start giving examples of useful or temptative virtual fonts.

2 A Dutch font

There are two specific features of Dutch typesetting, which can become automatic by changing the property list of DC fonts:

- the 'ij' ligature, in names like Eijkhout, Nijhof, Huijqen and in thousands of Dutch words;
- the fact that letter 'ë' becomes 'e' when the word is hyphenated just before this letter: conciërge will be hyphenated as conci-erge.

The automatization of the 'ij' ligature can be done very easily. We just need to add the lines

```
(LABEL C i)
(LIG C j H BC)
(STOP)
```

to the PL file (and the equivalent lines for the 'û' ligature (dont forget to set `\uccode"eb="cb`).

The problem of the disappearing dieresis is solved by a begin-of-word ligature: when the word is hyphenated, the letter 'ë' suddenly is at the beginning of the next line, and hence at the left boundary of the remainder of the word. If we set the rule *'ë' at the beginning of a word shall become 'e'*, then our problem is solved[6]. This can be written in the PL file as follows:

```
(LABEL BOUNDARYCHAR)
(/LIG H EB C e)
(/LIG H CB C E)
(STOP)
```

These additions will make any DC font behave 'the Dutch way'. Of course we will have to give these fonts new names, for example NLDC. But, since we haven't changed a single pixel of the original DC fonts, it would be very convenient if we could use PK files of DC fonts for our new fonts as well.

For this, we will convert our PL file into a VPL file: let's 'dutchify' font dcr10, by converting dcr10.pl into nldcr10.vpl:

Two new entries are necessary in the preamble:

```
(VTITLE Dutch DC font)
(MAPFONT D 0 (FONTNAME dcr10))
```

The FONTNAME field is the weak point of virtual fonts: the font name which appears in this field necessary depends on the operating system: for example if we want to use a DC Sans-Serif Bold Italic 10 points font, and write dcssbxti10 into that field, then the day when a MS-DOS[7] user wants to use our (compiled) virtual font, he/she will have a rather unpleasant surprise.

The MAPFONT command points to other fonts, it is followed by the internal number of each font. Since we take all of our characters from font dcr10, this is the only font we will define. It takes the internal number 0, which is the default font number. The Dutch font is ready.

Before we go any further, I would like to slightly change the subject and remind the reader that a **properly written file has a properly written header**, and by this I mean a header written using Nelson Beebes filehdr and Robert Solovays checksum specifications. Here is an example of such a header for the nldcr10.vpl file:

```
(COMMENT ********************************
 @Font-Property-List-file{
   author     = "Yannis Haralambous",
   version    = "alpha",
   date       = "28 March 1993",
   time       = "15:50:42 MET",
   filename   = "nldcr10.vpl",
   address    = "187, rue Nationale
                 59800 Lille
                 France",
   FAX        = "(33) 20.40.28.64",
   checksum   = "63003 1955 6452 40079",
   email      = "yannis@gat.citilille.fr",
   codetable  = "ISO/ASCII",
   supported  = "yes",
   docstring  = "Experimental virtual
                 property list file for
                 Dutch, based upon the
```

[4] The only solution would be to write raw PostScript code and put a white mask in front of the dot of 'j'; not a very clean solution, though.

[5] And one important conceptual difference: they contain the (system dependent) names of TFM files of the different fonts from which characters are taken. This makes VF fonts themselves system-dependent, and not only *their names* as in the case of "real" fonts!

[6] At least in most cases; this solution is not 100% clean: for reasons bound to the line-breaking algorithm of TEX, if for example we introduce a hyphen inside the word ('conci-ërge') and it happens to be hyphenated just after that hyphen, the dieresis will remain. . .

[7] MS-DOS is a computer operating system which appeared on planet Earth at the end of the twentieth century, and assumed that filenames would never require more than 8 letters of the Latin alphabet.

```
                dcr10 font."
}*************************************)
```

Nelson has developed a set of Emacs-LISP macros to create and update such headers automatically, inside GNU-Emacs. The checksum is calculated and verified by Robert Solovays `checksum` utility.

3 A font for Welsh

The Welsh language, spoken by some 250,000 people, is notorious for its long words (Don Knuth mentions Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch in the TEXbook) and for its multitude of accents: it needs the five standard vowels ('a', 'e', 'i', 'o', 'u'), and letters 'y' and 'w', *all* with acute, grave, dieresis and circumflex accent.

Note that all of these (both the letters and the accents) are present in the Cork encoding, what we need is their combination: to be able to hyphenate words (of the length of Dons example) with accented letters, these have to be included in the font as separate characters. We can always obtain the letter 'ŵ' by using the primitive `\accent`, but this will block the hyphenation process, which is of vital importance for this language.

So we'll have to define a variant of the Cork font, featuring in addition to the already existing 'a', 'e', 'i', 'o', 'u' with all four accents and 'ý', 'ÿ' also the letters 'ỳ', 'ŷ', 'ẃ', 'ẁ', 'ŵ', 'ẅ'. Let's give the variant of `dcr10`, the name `cydcr10` (`CY` is the ISO two-letter code for Welsh[8]).

The problem is slightly different than for the Dutch virtual font: once again we will remain in the same font, but this time we will have to combine characters: to get 'ŵ' we will combine a 'w' and a circumflex accent. Because of the width of the 'w' letter, the two signs must be superposed with a certain offset. How do we calculate this offset for every possible DC font?

The proper way to do it, would be by using METAFONT. We can ask the latter to write the exact offset between the letter 'w' and the circumflex accent in the `.log` file and then use that information. The advantage of this method is that the accent is always well positioned. The disadvantage is that this method is rather slow: for every font needed, one must run METAFONT and then edit the `VPL` file.

A different method is used by Eberhard Mattes' `QD-TeXVPL` (Quick and Dirty TEX to `VPL`). This method leaves TEX the task of placing the accent, using the `\accent` mechanism, and then collects the information. The disadvantage of this method is that the accent will always be placed in the middle and at the same relative height. But of course, this can be corrected manually by editing each `VPL` file.

Here is what to do: suppose we are placing letters 'ỳ', 'ŷ', 'ẃ', 'ẁ', 'ŵ', 'ẅ' at positions `0xA0`, `0xA7` – `0xA9`, `0xB3`, `0xB5` (and 'Ỳ', 'Ŷ', 'Ẃ', 'Ẁ', 'Ŵ', 'Ẅ' at positions `0x80`, `0x87` – `0x89`, `0x93`, `0x95`). We will prepare the following short TEX file:

```
\input qdtexvpl
\font\f=dcr10
\texvpl{^^80}{\f\`Y}
\texvpl{^^87}{\f\^Y}
\texvpl{^^88}{\f\'W}
\texvpl{^^89}{\f\`W}
\texvpl{^^93}{\f\^W}
\texvpl{^^95}{\f\"W}
\texvpl{^^a0}{\f\`y}
\texvpl{^^a7}{\f\^y}
\texvpl{^^a8}{\f\'w}
\texvpl{^^a9}{\f\`w}
\texvpl{^^b3}{\f\^w}
\texvpl{^^b5}{\f\"w}
\end
```

where we have one `\texvpl` command for each character: the first argument is its position and the second its description using plain TEX macros.

Let's call this file `welsh-wannabe.tex`. Next, we will run TEX on it and obtain `welsh-wannabe.dvi`. Then we will run `QDTeXVPL` with the following command line:

```
QDTeXVPL -d10.0 welsh-wannabe.dvi \
                     welsh-wannabe.vpl
```

Here is part of the resulting `VPL` file:

```
(CHARACTER O 200
    (CHARWD R 0.749817)
    (CHARHT R 0.939255)
    (CHARDP R 0.000000)
    (MAP
        (PUSH)
        (MOVERIGHT R 0.124969)
        (MOVEDOWN R -0.244980)
        (SETCHAR O 0)
        (POP)
        (SETCHAR C Y)))
```

We see the description of character `'200`, namely 'Ỳ'. The dimensions `CHARWD`, `CHARHT` and `CHARDP` are calculated out of the superposition of the boxes of the grave accent and of the letter 'Y' (Mattes puts both the grave accent and the 'Y' in a box and calculates the dimensions of this box).

The `MAP` field describes the character at position `O 200`. First we memorize the current position by a `PUSH` command. Then, we move 0.124969 em to the right, and 0.244980 em up. Next, we select font 16 and set character `0x00`, namely the grave accent. By a `POP` command we return to the position of the last `PUSH` (the usual `DVI` fifo stack) and typeset the letter 'Y' as if nothing happened.

[8] This is by no means the announcement of a Welsh font encoding standard. A subgroup of the Technical Working Group on Multiple Language Coordination will take care of that: the new font encoding will be part of the Welsh TLP (TEX Language Package).

Now we have to merge this file with the `PL` file of the `dcr10` font and call the resulting file `cydcr10.vpl`. Don't forget the `VTITLE` and `MAPFONT` fields, similar to those of the Dutch font. (Don't forget Nelsons file header!) By making some cut-and-paste[9] operations we replace the previous character metrics by the new descriptions.

Are we finished? Certainly not. A font is more than just a bunch of characters. We must also take a look at the relationships between these characters, namely ligatures and kernings. We have to check that the previous characters were not involved in some ligatures (that's easy with DC fonts which have only a dozen ligatures, but less trivial with Arabic fonts which have about 5,000 of them).

And then we will have to generate kerning pairs for the new characters: it is logical that in most cases 'ŵ' should be kerned exactly like 'w', and so on. To automate this procedure, the author has written an utility called `AdjKerns`, which generates kerning pairs for a font out of existing pairs and a set of rules (for example 'œ' should be kerned like 'o' on the left and like 'e' on the right).

Here is how to proceed: first we will write a small file with the rules (this file depends only on the encoding), let's call it `welsh-krn-rules.kcf` (`kcf` for kerning configuration file). Its contents will be:

```
KERN H 80 LIKE C Y
KERN H 87 LIKE C Y
KERN H 88 LIKE C W
KERN H 89 LIKE C W
KERN H 93 LIKE C W
KERN H 95 LIKE C W
KERN H A0 LIKE C y
KERN H A7 LIKE C y
KERN H A8 LIKE C w
KERN H A9 LIKE C w
KERN H B3 LIKE C w
KERN H B5 LIKE C w
```

By executing the program with the following command line:

```
AdjKerns -c welsh-krn-rules.kcf \
                          cydcr10.vpl
```

it will read all kerning pairs and ligatures, disregard those of the previous characters, generate new pairs according to the rules in `welsh-krn-rules.kcf` and overwrite the file `cydcr10.vpl`. The latter will be the `VPL` of our newborn Welsh font.

4 PostScript "Expert" Fonts

When Don Knuth wrote the Computer Modern fonts, he found it very natural to include the 'ff', 'ffi' and 'ffl' ligatures in the standard font encoding, and to make a separate font for small caps. Unfortunately the makers of PostScript fonts have decided otherwise: (1) 'ff', 'ffi' and 'ffl' were left outside, (2) small caps are sometimes available, but in an auxiliary font *which does not include uppercase letters!*

Virtual fonts allow us TeX-users to use PostScript fonts without renouncing to our habits and use both the 'f-ligatures' and small caps with the same ease as we use DC fonts.

Let us resume the situation, by taking a concrete example: the beautiful Centaur font (drawn by Bruce Rogers in 1928), commercialized by Adobe Systems:

- the plain PostScript font `CentaurMT` (`MT` for MonoType) contains the usual set of characters (uppercase and lowercase), including 'ﬁ' and 'ﬂ' ligatures;
- the 'expert' PostScript font `CentaurExpertMT`, contains the ' ', ' ' and ' ' ligatures, small caps , as well as olstyle digits . It does not contain uppercase letters.

And here is what we want:

- a plain font `centaur` with uppercase, lowercase letters and the five 'f-ligatures': ' ', ﬁ', 'ﬂ, ' ' and ' ';
- a small caps font `centaursc` with uppercase letters, small capitals, and no 'f-ligatures'. Eventually we could put the oldstyle digits in this font.

In this way it will be straightforward to write « une ﬁlle ﬂamande a igée mais a reusement ra née » or „ B B , B , B “.

The reader has already guessed the solution: these fonts will be virtual, and will each one take characters from both PostScript fonts. The `TFM` files of the original PostScript fonts will have names starting by an 'r' (raw): `rcentaur` and `rcentaurexp`. Both `centaur` and `centaursc` will use characters from fonts `rcentaur` and `rcentaurexp`.

There is an utility to do this job automatically: Vulcano, written in 1992 by the author. Vulcano will read both `AFM` files and according to a certain configuration file will use the information to write a `VPL` file for a plain or a small caps font.

The configuration files are called `dc.vcf` and `dc-sc.vcf` (`VCF` for 'Vulcano Configuration File'). Here is the command line (splitted in two lines) for making the plain font:

```
Vulcano -c dc.vcf -p rcentaur.pl \
 -v centaur.vpl  -a CentaExpMT.afm \
        rcentaurexp.pl CentaMT.afm
```

The `-c` option is used to indicate the configuration file, `-p` gives the name of the `PL` file of the raw PostScript font, `-v` the name of the `VPL` file we wish to have, `-a` is used to add supplementary `AFM` files (we can use `-a` up to 255 times: theoretically we can take every

[9] Emacsians say "kill-and-yank"...

character from a different PostScript font…), the two arguments of `-a` are the `AFM` file name and the name of the resulting raw `PL` file.

To obtain the small caps font we just have to modify the command line a bit:

```
Vulcano -c dc-sc.vcf -p rcentaur.pl \
 -v centaursc.vpl -a CentaExpMT.afm \
        rcentaurexp.pl CentaMT.afm
```

The reader will notice that the arguments of `-p` and `-v` haven't changed: that is natural since the information Vulcano gathers out of the `AFM` files is exactly the same, and hence the resulting raw `PL` files will be the same for both runs.

It is very easy to make new `VCF` files. Let's take a look to their syntax. In PostScript, every glyph has a name; the 'encoding' actually affects character positions to these names. The first part of a `VCF` file consists of a list of PostScript names for glyphs appearing in the font, followed by the various positions they occupy in the specific encoding:

```
START Encoding DC (YH 26mar93)
grave -> 0
acute -> 1
circumflex -> 2
tilde -> 3
...
udieresis -> 252
yacute -> 253
thorn -> 254
germandbls -> 255
STOP
```

The second part of a `VCF` file, concerns the eventual boundary character, left boundary ligatures and ordinary ligatures:

```
START Ligatures 14
21 45 =: 22
27 105 =: 30
27 108 =: 31
33 96 =: 189
39 39 =: 17
44 44 =: 18
45 45 =: 21
60 60 =: 19
62 62 =: 20
63 96 =: 190
96 96 =: 16
102 102 =: 27
102 105 =: 28
102 108 =: 29
STOP
END
```

The use of glyph names in this second part would lead to inconsistencies, here is an example: the glyph `hyphen` is used in two cases in the DC font, at position `45` for the word separator, and at position `127` as the line-break-hyphen. As we can see in the list, a ligature is provided for character `45` but *not* for character `127`; this distinction would not be possible on the PostScript glyph name level.

And now let's take a look at the `VPL` files created by Vulcano; this time we have two fonts to combine:

```
(MAPFONT D 0 (FONTNAME rcentaur))
(MAPFONT D 1 (FONTNAME rcentaurexp))
```

Here is the code for characters ‘ ’ and ‘fi’:

```
(CHARACTER D 27 (MAP (SELECTFONT D 1)(SETCHAR D 86))(COMMENT PS: ff)
   (CHARWD R 552)
   (CHARHT R 673)
   (CHARDP R 1)
   (COMMENT
      (LIG i -> ffi)
      (LIG l -> ffl)
      )
   )
(CHARACTER D 28 (MAP (SETCHAR D 174))(COMMENT PS: fi)
   (CHARWD R 479)
   (CHARHT R 674)
   (CHARDP R 8)
   )
```

In the first case we switch to font 1 (`rcentaurexp`) and select character `86`, in the second case we remain in font 0 (the default font `rcentaur`) and take character `174`.

5 How to Get Rid of Virtual Fonts

If you are sending your file to a correspondant who does not have the same virtual fonts as you (for example if you are using a Dutch or Welsh virtual font and want to switch back to the original DC fonts), then there is an utility for 'de-virtualizing' your `DVI` file.

This utility, called `DVIcopy` and written by Peter Breitenlohner, will replace each virtual character by its real extension. It is a 'must' if you plan to have your own personalized virtual fonts.

6 Availability

All mentionned tools are in the public domain. They are written in ANSI C and can be found on various servers: `QDTeXVPL` can be found at Stuttgart (IP `129.69.1.12`) in directory

`/soft/tex/fonts/utilities/qdtexvpl`

while `Vulcano` and `AdjKerns` are kept in Paris (IP `129.199.104.3`) in `/pub/tex/yannis`. Nelson Beebes `filehdr` package as well as Robert Solovays `checksum` can be found in Salt Lake City (IP `128.110.198.2`), in `/pub/tex/bib` and `/pub/tex/pub`.

`DVIcopy` is written in Pascal `WEB` (a C version of it can be obtained through Tom Rokickis `WEB2C` translator). It can be fetched from Stuttgart, directory `/soft/tex/systems/pc/utilities`.

7 Conclusion

The examples in this paper should not be considered as effective TEXware, but merely as hints on areas open to further development. Many people have written tools to create virtual fonts: Jiří Zlatuška (`ACCENTS`), Alan Jeffrey (tools in `AWK`) and others.

The author hopes that his effort in convincing the reader that virtual fonts are *not* for wizards only, has succeeded. So, have fun!

The Birth of a Virtual Font
The AdjKerns Utility

Yannis Haralambous

yannis@gat.citilille.fr

1 Introduction

Making a baby can be a difficult process: you need at least two people (with the adequate tools) you need some favorable circumstances, and you need motivation.

What about virtual fonts? well, we have motivation enough: several languages are not covered by DC fonts, in particular Welsh, Esperanto, Maltese, many African languages (the glyphs for the latter are included in Joerg Knappens FC fonts) etc. These languages need virtual fonts to be typeset. The circumstances are favorable enough. Up to now, though, we were missing one tool.

Here is the problem: you want to substitute a character in the DC font by some special character you need in your target language. OK. How to construct the virtual representation for this character? well, for that we have Eberhard Mattes' `QDTeXVPL`. It will give us the description of all characters we need. Now suppose you replace an 'A' with some accent, by an 'E' with some other accent. What about all the kerning pairs involving that 'A' you took away? And what all those new kerning pairs you need for the 'E'?

Do you want to replace the old kerning pairs by new ones, by hand?, well, you can try; but if you ever attempt to take a coffee break, then you will never be able to recognize again what you have changed, and what remains to be changed. Not to mention the nasty habit of PL files of having kerning pairs applying to more than one characters. This optimizes parsing speed, but surely makes it more difficult to make manual changes.

The utility I'm presenting does all this automatically. It will read a `[V]PL` file and will

- remove all kerning pairs involving removed characters;
- introduce kerning pairs for new characters;
- add or remove ligatures.

To illustrate the use of this program, I provide a real-world example, namely a virtual font for Esperanto. You will follow the whole process of giving birth to this font, starting by its mother: the common `dcr10` font.

2 How it works

`AdjKerns` has been written in ANSI C[1]. It should run everywhere; it has been tested on a Macintosh (MPW 3.2 and Think C 5.0).

The command line should look like this:

```
AdjKerns [-c foo.kcf] \
     [-o <something>]  <input file>
```

The input file is a `PL` or a `VPL` file[2]. Out of this file `AdjKerns` will read only the `LIGTABLE`, the rest will remain untouched.

The rules on how to modify the `LIGTABLE` are read from the configuration file `foo.kcf`. This file is optional; if you don't include any rules, then `AdjKerns` will just rewrite your `[V]PL` file, by ignoring any optimization attempts of the `LIGTABLE`. The file will be bigger, but the result will be the same; it will be easier to modify, though.

The `-o` option specifies the output file. If no such is specified, then the input file is overwritten (you don't need that file anyway…).

Let's see now the syntax of the `KCF` file. The following lines are allowed:

```
KERN <char> LIKE <char>
KERN <char> LEFT LIKE <char>
KERN <char> RIGHT LIKE <char>
KERN <char> LEFT LIKE <char>
                RIGHT LIKE <char>
UNKERN <char>
ADD <lig> <char> <char> -> <char>
ADD <lig> BOUNDARYCHAR <char> ->
                          <char>
REMOVE <lig> <char> <char> -> <char>
REMOVE <lig> BOUNDARYCHAR <char> ->
                           <char>
% comments, like in TeX
```

where `<char>` means (like in a `[V]PL` file, either `C` followed by a character, or `D` followed by a decimal number, or `O` followed by an octal number, or `H` followed by a hexadecimal number. `<lig>` means one

[1] Be aware! its author is a lousy C programmer, so *don't* trust the program as you trust TeX. Keep a critical eye for possible bugs, and report them!

[2] If you look closer you'll see that there is no big difference; the latter has a few more possibilities, otherwise the syntax is the same.