Ogonkify

Juliusz Chroboczek

Short Contents

Table of Contents

The Ogonkify package

The Ogonkify package contains two programs: a utility for adding composite characters to fonts in a semi-automatic fashion, and a program for converting PS output — notably Netscape and Mosaic output — to use these fonts.

1 The McKornik Jr. Public License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

In particular, this program is provided as is, without any warranty, not even of any kind. It might (or might not) do what it is supposed to do, it might (or might not) destroy your printer, and it might (or might not) be useful to you. In any case, you are on your own.

This program is written in Perl. While stronger people find reading Perl code character-building, it should not be shown to people in their formative years. The author will not accept any responsibility for any moral grief caused.

I would like to ask you *not* to distribute the generated fonts without including a pointer to the original AFMs and the rest of the code. Furthermore, please notify me if you decide to include this code in a larger piece of software.

If you find this software useful, local goodies, makowiec, beer from Zywiec, postcards from exotic places and all sorts of things are accepted.

2 Using Ogonkify

This chapter describes the various uses of Ogonkify.

2.1 Viewing with Ghostscript

Due to a bug in some versions of ghostscript, in order to view the output of Ogonkify you may need to run ghostscript with the `-dNOPLATFONTS` option, as in

```
gs -dNOPLATFONTS foo.ps
```

or in

```
ghostview -arguments -dNOPLATFONTS foo.ps
```

See the `gs(1)` and `ghostview(1)` manual pages for more details.

2.2 Printing from Standard Applications

The main use of Ogonkify is to allow various standard applications to print in various languages. As of this writing, the `ogonkify` script knows how to handle Netscape, Mosaic, the mail printer MP, StarOffice, and ApplixWare. Its usage is described in full detail in the `ogonkify(1)` manual page.

2.3 Printing with a2ps

`a2ps` is a PostScript formatter with pretty-printing capabilities. Starting with version 4.7.21, it automatically uses Ogonkify, which is distributed with it. See Section “a2ps manual” in *a2ps manual*.

`a2ps` can be obtained from:

```
<URL:http://www-stud.enst.fr/~demaille/a2ps.html>
```

2.4 Printing with GNU Enscript

GNU Enscript is a program for printing text files on PostScript printers written by Markku Rossi and distributed with other GNU software. You will find it on any mirror carrying GNU software.

GNU Enscript is one of the few programs that allow the user to select the encoding of the files he wants to print, and correctly reencoding the fonts it uses . In order to obtain information about the characters available in a font, it uses AFM files.

If you are only interested in printing in Courier, you might want to simply use the IBM Courier fonts and the AFM files provided with them. However, you may also want to use the output of Ogonkify; for printing in Times and Helvetica, this is the only solution.

Here are installation procedures for using the fonts generated by the `instogonki` script with `genscript`. They assume that `genscript` is properly installed, and that you know which `genscript.cfg` file it reads.

1. In the directory `$PFADIR`, type

   ```
   % mkafmmap *.afm
   ```

   (the `mkafmmap` program is installed with GNU Enscript).
2. In the `genscript.cfg` file, add your `$PFADIR` directory to the AFMPath.

You can now print with the composite characters. Example:

```
% genscript -X latin2 -f Times-Roman-Ogonki@12 \
            -o wierszyk.ps wierszyk.txt
```

or

```
% genscript -X latin2 -f Helvetica-Ogonki@12 \
            -o wierszyk.ps wierszyk.txt
```

3 Adding new characters

This chapter attempts to explain how to extend Ogonkify to handle supplementary characters.

3.1 Background

A typical Type 1 font for the latin alphabet contains "pure" characters, such a `A` or `acute`, and composite characters, such as `Aacute`, which are composed of the `A` character, and the `acute` character using the Type 1 `seac` operator.

Unfortunately, most fonts do not contain all the characters in the ISO Latin-2 character set. In particular, most of the Polish characters (with the exception of `oacute` and `Oacute`) are usually missing (a notable exception is IBM Courier — not Adobe Courier — which contains many useful glyphs). However, the components of those glyphs are present. It should not be difficult to add the characters we need if the necessary tools were available... At least four methods could be used:

i) Do not change the original font, but do overstriking for individual characters (e.g. typeset an `a`, then move backwards and typeset an `ogonek` for `aogonek`). This method is widely used, for instance by TeX.

ii) Parse the `PFA` or `PFB` file containing the font program for a Type 1 font, and generate a new `PFA` containing the composite characters. This should not be too difficult (`PFA` parsers are freely available), but would require that users have the `PFA` files corresponding to the fonts they use, which is often not the case. Furthermore, I believe that it would violate the license of the fonts. I am not quite sure what to make of the following:

> Adobe Systems' Type 1 font programs are licensed for use on one or more devices (depending on the terms of particular licenses). These licenses would permit the use of a licensed program in a system that translates a Type 1 font program to some other format in the process of rendering, as long as a copy of the program (even in translated form) is not produced.
>
> *Adobe Type 1 Font Format*, p. 7, Adobe Systems Inc.

iii) Download the original font, and add to its Charstrings dictionary the supplementary characters using PostScript. This would not violate the license agreement as the modified font would exist only in the printer. However, many PostScript interpreters do not allow tampering with Type 1 font dictionaries. Copy protection, as always, only bothers honest users.

iv) Create a new Type 3 font dictionary which draws characters by using the characters in the original font. This has the benefits of working and being legal. I expected it to be quite inefficient, but found it to be reasonably fast.

The `composite.ps` file contained in this distribution, and the accompanying perl program `composeglyphs`, follow scheme (iv). The following sections describe them in more detail, including information on extending the code to create other composite characters.

> **Please note:** our usage of the word "composite" has noting to do with Adobe's notion of "composite fonts". This is a bug.
>
> **Please note:** a Polish typographer would be appalled to see that we consider the "ogonek" as a diacritical mark, and thus harm the integrity of the two letters that all Poles love. Indeed, in a proper Polish font, the tail of `aogonek` would have a different shape than that of `eogonek`. Considering however the poor availability of fonts with the needed characters, we do not currently have the luxury to whine about such esthetic problems.

3.2 Non-orthodox uses of Adobe Font Metrics files

Information about the fonts comes from *Adobe Font Metrics* (AFM) files. Much more information is in AFMs then usually known; in particular, the encoding vector can be derived from the AFM, and AFMs contain the composite character information. Therefore, a program could generate the needed font — encoding vector and everything included — from the AFMs.

In order to simplify the handling of AFM information, we use three different AFMs on every run of the program: (i) an AFM with the encoding vector to use, (ii) the AFM of the original font, and (iii) an AFM with supplementary composite character information. The latter will usually have to be supplied by the user. It is needed because AFMs provided with Type 1 fonts usually only contain composite information about the characters already in the font (in fact, the Adobe documentation does not make it quite clear whether it is legal to insert composite information about characters not already in the font into an AFM).

3.3 Fonts with composite characters

The generated font contains (i) the characters that were in the original font, and (ii) the characters which were not in the original font but for which composite information was provided. AFM files for the generated fonts are generated too, which means that the fonts can be used with most applications. Furthermore, they can be reencoded, just like any well-behaved PostScript font.

In order to maintain compatibility with PostScript Level 1, every composed font must be based on a base encoding vector, and characters can only be composed from components in that vector. Thus, in order to build `Nacute` from `N` and `acute`, both `N` and `acute` must be in the base vector.

However, as the base vector is only ever used internally, much liberty can be taken when designing it. In particular, it doesn't need to be compatible with any other vector, and there

is no reason to avoid the control character range. The base encoding vector that I use is called `OgonkiEncoding`; it is based on Latin-2, but contains all the characters of `StandardEncoding` (although in strange places).

There is a limitation on the composite information that can be used. In particular, the only composite entries used are of the form:

```
CC ... 2 ; PCC x 0 0 ; PCC ' ... ; ...
```

In plain words, there must be exactly two characters to compose, the first character must be set at the origin, and the width of the composite character is taken to be that of the first character. This limitation is easy to lift, and I will generalize the code if you send me AFMs that do not obey this convention (I have never seen any, and, indeed, this is the format required by the Type 1 `seac` operator). AFMs that do not obey it are (hopefully) gracefully handled (you should see warnings about `CC` entries being ignored).

3.4 Usage of the composite script

The `composeglyphs` script can be run to generate either an encoding vector or a new font. It is a perl script, and only reformats the data; the real magic is in the file `composite.ps`.

In order to generate an encoding vector from a suitable AFM, `composeglyphs` is run as follows:

```
% composeglyphs -e latin2.afm -E latin2.enc
```

which will generate the file `latin2.enc` from the AFM file `latin2.afm` (this is the default). Any AFM file is suitable as input, but most AFMs do not contain all the possible characters of an encoding vector (the missing ones will be replaced by `.notdef`).

In order to generate a font program, more input must be provided:

```
% composeglyphs -i ptmr.afm -c ptmr-c.afm \
                 -o ptmr-o.ps -n Times-Roman-Ogonki \
                 -a ptmr-o.afm \
                 -e ogonki.afm -t adobe.afm
```

where `-i` specifies the AFM of the original font, `-c` supplementary composite character information, `-o` the font program to generate, `-n` the name of the new font, `-a` the name of the AFM file to generate, `-e` the base encoding AFM to use, and `-t` the target encoding.

The file `makecomp` contains a shell script (run by the installation program) which generates fonts with composite characters for the Times, Courier and Helvetica families from the supplementary AFMs `*-c.afm`.

3.5 A step-by-step example

Assume that you want to add the `ccaron` and `Ccaron` characters to the Times-Roman font. Start from similar characters already present in the base Times-Roman font — for example, `zcaron` and `Zcaron`. Take the corresponding *composite character* (`CC`) line from the `ptmr.afm` file:

```
CC zcaron 2 ; PCC z 0 0 ; PCC caron 55 0 ;
```

and insert it into the `ptmr-c.afm` file, changing the names of the characters thus:

```
CC ccaron 2 ; PCC c 0 0 ; PCC caron 55 0 ;
```

This line can be used as a starting point. Execute `./instogonki`, typeset some text with the new character, and fine-tune the last set of coordinates (`55 0`).

If you create new AFMs or modify the existing ones, please send me a copy so that I can include them in the distribution.

4 Reaching the author

This manual and most of the Ogonkify package were written by Juliusz Chroboczek, <jec@dcs.ed.ac.uk>.

My current snail-mail address is:

Juliusz Chroboczek
Department of Computer Science
Mayfield Road
Edinburgh EH9 3JZ
Scotland, United Kingdom
tel. +44/131/650-5163
fax. +44/131/667-7209