# Outline font extensions for Arabic typesetting

Karel Píška
Institute of Physics, Academy of Sciences
182 21 Prague, Czech Republic
piska (at) fzu dot cz

**Abstract**

The contribution demonstrates applications of the programs FontForge (by George Williams) and MetaType1 (by Bogusław Jackowski et al.) for development and maintenance of outline versions of Arabic fonts and shows tools for glyph transformations. Generating of stretchable glyphs is also discussed. Building of Type 1 fonts for Arabic typesetting with MetaType1 ("LM-ization") is under development process.

## 1 Introduction

The current text describes selected "technical techniques" of creating, modifying and maintaining outline fonts for use with Arabic typography.

We start with some font adaptations and modifications using FontForge for relatively simple glyph transformations. Executing more complex changes would be inefficient.

The next examples demonstrate representation of glyphs in MetaType1 (in fact, it is METAPOST), font modifications and some results of Type 1 fonts dynamically generated by MetaType1.

## 2 Font transformations with FontForge

The open source font editor FontForge [15], developed by George Williams, contains many commands for the creation and modification of fonts in numerous standard formats. Along with its interactive facilities, FontForge has a scripting language which allows automatic batch processing. Thus, existing fonts, e.g., Computer Modern Type 1 mathematical fonts, can be adapted into fonts oriented to Arabic presentation by executing appropriate glyph transformation commands. The example in fig. 1 demonstrates how to flip (reverse, mirror) a symbol to achieve the effect described in the paper on 'Dynamic Arabic Mathematical Fonts' ([11], fig. 1) in our Type 1 representation. The original symbol (left) has been flipped horizontally (middle) and then vertically (right):

```
#!/usr/local/bin/fontforge
Open("cmex10.pfb");
 Select("summationtext");
 HFlip(CharInfo("Width")/2); VFlip();
 CorrectDirection();
 SetFontNames("amcmex10",\
 "ArabicComputerModern",\
 "ArabicMathCMEX10");
Generate("amcmex10.pfb","",0x240000);
# no flex hints, hints, round,
# no afm, no tfm
```

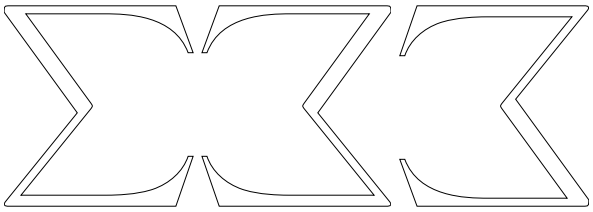

**Figure 1**: Flip twice or rotate.

The identical transformation could be done by rotation:

```
 Select("summationtext");
 Rotate(180);
```

## 3 Font development with MetaType1

MetaType1 [6], developed by Bogusław Jackowski, Janusz M. Nowacki, and Piotr Strzelczyk, is a METAPOST-based package for producing, auditing, enhancing and otherwise handling outline PostScript fonts in the Type 1 format. One part of the package is a converter from Type 1 to MetaType1 source. A practical approach to maintaining a font with MetaType1 is to start from an existing Type 1 font or from a font just converted into Type 1 (e.g., from METAFONT sources). In the present case for Arabic we can and *want* to use as a base (for further modifications, extensions, etc.) the `xnsh14` font in the Naskhi style available from the ArabTEX distribution [9].

The most important ideas of the MetaType1 package are:

- programmable font description in source form, (the language is METAPOST with extensions for font support);
- the glyphs are defined in their outline representation (the recommended approach);
- very simple definition of composite glyphs, especially glyphs with accents, which is significant for all Latin fonts;
- automatic generation of glyph and metric files (`pfb, afm, tfm, pfm`) from scratch;
- the glyphs are (may be and must be) denoted by (PostScript) names, therefore the number of glyphs is not limited, although the Type 1, `tfm` output and encodings are restricted (no more than 256 encoded characters available);
- the glyph definitions also contain metric data: dimensions (width, height, depth, italic correction), ligatures, kerning pairs and other information, to allow for easier further conversion into OpenType or Type 3.

These features have been applied during the development of the Latin Modern collection [7] and other fonts produced by the authors. Similar "LM-ization" could be executed for Arabic because there is no fundamental difference between Latin accents and Arabic diacritic marks.

The following example and fig. 2 illustrate producing composite Arabic glyphs with MetaType1.

```
def mark_down(text glyh_acc_,glyh_,acc_) =
standard_introduce(glyh_acc_);
beginglyph(glyh_acc_);
 use_glyph(glyh_);
 use_glyph(acc_) % offset of the accent =
 (round((wd.uni_name(glyh_)-wd.uni_name(acc_))/2),
 dp.uni_name(glyh_)-ht.uni_name(acc_));
% recalculation of metrics
 wd.uni_name(glyh_acc_)=wd.uni_name(glyh_);
 ht.uni_name(glyh_acc_)=ht.uni_name(glyh_);
 dp.uni_name(glyh_acc_)=dp.uni_name(glyh_)
 -ht.uni_name(acc_)+dp.uni_name(acc_);
 fix_hsbw(wd.uni_name(glyh_acc_),0,0);
endglyph;
enddef;

mark_down("bah")("bah_s")("one_dot_down");
mark_down("pah")("bah_s")("three_dots_down");
% definition of "mark_up" macro is similar
mark_up("tah")("bah_s")("two_dots_up");
```

The next example shows an excerpt from the Type 1 representation in the readable form disassembled by `t1disasm` from the `t1utils` package. The dot mark in `nun` is omitted.

```
/nun.fin {
        0 433 hsbw
        -278 70 hstem
        0 71 hstem
        0 35 vstem
```

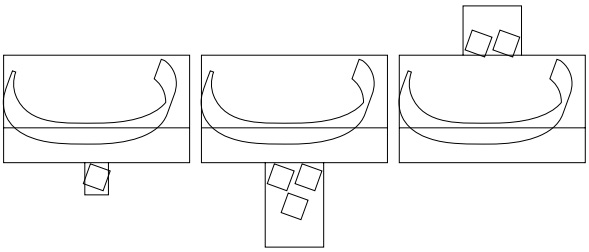

**Figure 2**: Composite glyphs: b — p — t.

```
        356 36 vstem
        451 67 rmoveto
        -11 4 rlineto
        -27 0 -19 13 -22 22 rrcurveto
        -11 4 rlineto
        -24 -67 rlineto
        13 -38 6 -40 0 -40 rrcurveto
        0 -13 -1 -12 -2 -13 rrcurveto
        -49 -84 -78 -11 -34 0 rrcurveto
[...]
        closepath
        endchar
} ND
```

The result of conversion into the METAPOST/MetaType1 representation (in an absolute coordinate system!) is:

```
beginglyph(_nun.fin);
 save p; path p[];

 z0 0=(451,67);
 z0 1=(440,71); z0 1a=(413,71); z0 2b=(394,84);
 z0 2=(372,106);
 z0 3=(361,110);
 z0 4=(337,43); z0 4a=(350,5); z0 5b=(356,-35);
 z0 5=(356,-75); z0 5a=(356,-88); z0 6b=(355,-100);
 z0 6=(353,-113); z0 6a=(304,-197); z0 7b=(226,-208);
[...]
 z0 21=(426,0);
 p0=compose_path.z0(21); Fill p0;
 fix_hstem(71)(p0) candidate_list(y)(0, 71);
[...]
 standard_exact_hsbw("nun.fin");
endglyph;
```

And my subsequent conversion of the path definition into relative coordinates, as it will be necessary to eliminate the dependence on absolute coordinate values:

```
def nunf_z(suffix nz) =
z.nz 0=(451,67);
z.nz 1=z.nz 0+(-11,4);
z.nz 1a=z.nz 1+(-28,0); z.nz 2b=z.nz 1a+(-17,12);
z.nz 2=z.nz 2b+(-23,23);
[...]
z.nz 7=z.nz 7b+(-34,0);
z.nz 7a=z.nz 7+(-61,0);
z.nz 8b=z.nz 7a+(-70,35);
[...]
z.nz 15=z.nz 15b+(111,0);
z.nz 15a=z.nz 15+(111,0);
```

```
[...]
z.nz 22=z.nz 0;
enddef;
```

This path has been slightly modified and will be further adapted in an extended stretchable definition of the letter `nun` in the final position (without the dot) — see later.

The METAPOST macro definitions may define glyphs or their parts, and we can use them to describe glyphs already present in a font or to compose modified or new glyphs.

The transformations we saw in fig. 1 can be expressed in METAPOST to produce the same results:

```
numeric l,d; l:=wd._summationtext;
 d=dp._summationtext;

%  Horizontal Flip:
p0=compose_path.z0(22)
 reflectedabout((l/2,0),(l/2,1));
correct_path_directions(p0)(p);

%  Horizontal and Vertical Flip:
p1=compose_path.z0(22)
 reflectedabout((l/2,0),(l/2,1))
 reflectedabout((0,d/2),(1,d/2));
correct_path_directions(p1)(p);

%  Rotation
p2=compose_path.z0(22)
 rotated 180 shifted (l,d);
```

## 4 Glyph stretching with MetaType1

Azzeddine Lazrek et al., in the papers about typesetting Arabic (RyDArab [12] and CurExt packages [10] and dynamic fonts [11]), describe the use of dynamic Type 3 fonts and corresponding `tfm`, `map` and `enc` files for generating final PostScript documents with (LA)TEX and `dvips`. The supported version of MetaType1 supports producing only Type 1 fonts, and Type 1 (and also metric) files cannot be dynamic.

But `tfm` (as the RyDArab system does) and Type 1 can be generated dynamically. As the next consecutive step after generating metrics we can produce (dynamically with MetaType1) the Type 1 font corresponding to the equivalent Type 3 font and substituting it.

The variable-width kashida is demonstrated in fig. 3. The Type 3 commands from [11] were rewritten into METAPOST macros giving the similar commands in Type 1 where, of course, each glyph should have its own charstring definition. Glyphs, like metrics, can be generated on the fly for a given width.

**Figure 3**: Stretchable kashida in Type 1.

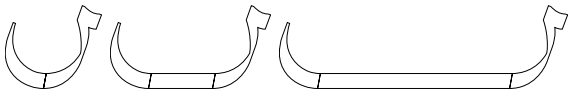

**Figure 4**: Stretchable glyph with an hrule filler.

Fig. 4 shows a primitive glyph elongation where only a horizontal rule as the filler is inserted and the right and left part of the glyph `nun` in the final position (without dot) are shifted.

The following METAPOST commands and fig. 5 demonstrate a more sophisticated elongation algorithm: the parameter *addwx* changes control points and control vectors. The solution was inspired by methods described by Daniel M. Berry [1]. Here we have decided to preserve glyph heights and also their right and left parts. Probably more complex outline contour curves could be defined.

```
def nunf_z(suffix nz)(expr addwx) =
addwxa:=round(addwx/2);
z.nz 0=(451,67)+(addwx,0);
z.nz 1=z.nz 0+(-11,4);
z.nz 1a=z.nz 1+(-28,0); z.nz 2b=z.nz 1a+(-17,12);
[...]
z.nz 7=z.nz 7b+(-34,0)-(addwxa,0);
z.nz 7a=z.nz 7+(-61,0)-(addwxa,0);
z.nz 8b=z.nz 7a+(-70,35);
[...]
z.nz 15=z.nz 15b+(111,0)+(addwxa,0);
z.nz 15a=z.nz 15+(111,0)+(addwxa,0);
[...]
z.nz 22=z.nz 0;
enddef;

def nun_fin_v(suffix code)(expr addx) =
 standard_introduce("nun.fin_v" & decimal(code));
 wd._nun.fin_v.code:=wd._nun.fin+addx;
 ht._nun.fin_v.code:=ht._nun.fin;
 dp._nun.fin_v.code:=dp._nun.fin;
 beginglyph(_nun.fin_v.code);
  save p; path p[];
  nunf_z(0)(addx); p0=compose_path.z0(21);
  Fill p0;
  standard_exact_hsbw("nun.fin_v" & decimal(code));
 endglyph;
enddef;
nun_fin_v(300)(+300);
```

**Figure 5**: Stretchable glyph.

D. Berry [1] and A. Lazrek [11] propose to use PostScript Type 3 fonts and W. Bzyl [2, 3] reintroduced Type 3 fonts and showed how to extend the MetaType1 package to produce Type 3 (but it did not find the support in recent MetaType1 distributions). For me the future of Type 3 fonts is not clear, screen rendering algorithms for Type 3 are worse than for other font formats, probably nobody is going to improve them, and I do not include Type 3 in my contribution.

## 5 Conclusion

I have no detailed information about commercial and copyrighted products of Thomas Milo [13] (and I do not have these products). I expect the development and use of Arabic fonts will be discussed with authors of packages for multilingual typesetting including Arabic: Y. Haralambous [14], H. Fahmy [4], K. Lagally [9], J. Kew [8], A. Lazrek, and others. I have no support for integrating dynamically generated or stretchable fonts into TEX. Some new line breaking and justification algorithms could be developed, for example, to spread the word box to a specified width and then to generate dynamically the appropriate glyph instance by demand on the fly to composite a compound "ligature". Or to produce glyphs only in a restricted set of point sizes and apply some variant of a micro-typographic alignment or justification as in pdfTEX [5].

This approach uses "small" (max. 256 glyphs) `tfm` and `pfb` files. We could convert them into OpenType (as the LM fonts have been converted). But I do not know: "Could we integrate Type 3 into OpenType?" or "Is it possible to create dynamic OpenType?"

A limitation to one direction, the current trend towards a single, huge and static outline OpenType font file (for each typeface in important point sizes), may not be wise. It's unlikely this will be the final termination point of font development, and thus will not be the best solution in the future development of computer font technology.

## References


[1] Daniel M. Berry. Stretching Letter and Slanted-baseline Formatting for Arabic, Hebrew, and Persian with ditroff/ffortid and Dynamic PostScript Fonts. *Software—Practice & Experience*, 29(15), 1417–1457, 1999.

[2] Włodzimierz Bzyl. Reintroducing Type 3 fonts to the world of TEX. *Proceedings of the XII European TEX Conference*, pp. 219–243, Kerkrade, the Netherlands, 23–27 September 2001, 2001.

[3] Włodzimierz Bzyl. The Tao of Fonts, *TUGboat* 23(1):27–40, 2002.

[4] Hossam A. H. Fahmy. AlQalam for typesetting traditional Arabic texts. In this volume, pp. 159–166.

[5] Hàn Thế Thành. Micro-typographic extensions to the TEX typesetting system. *TUGboat* 21(4), 317–434, 2000.

[6] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk. Programming PostScript Type 1 Fonts Using MetaType1: Auditing, Enhancing, Creating. *EuroTEX 2003 Proceedings*, *TUGboat* 24(3):575–581, 2003; `ftp://bop.eps.gda.pl/pub/metatype1`.

[7] Bogusław Jackowski, Janusz M. Nowacki. Enhancing Computer Modern with accents, accents, accents. *TUGboat* 24(1):64–74, 2003; `CTAN:/fonts/lm`.

[8] Jonathan Kew. XETEX, the Multilingual Lion: TEX meets Unicode and smart font technologies. *TUG 2005 Conference Proceedings*, *TUGboat* 26(2):115–124, 2005.

[9] Klaus Lagally. ArabTEX — Typesetting Arabic with vowels and ligatures. *EuroTEX 92: Proceedings of the 7th European TEX Conference*, ed. J. Zlatuška, pp. 152–172, Brno, Czechoslovakia, 1992.

[10] Azzeddine Lazrek. CurExt, typesetting variable-sized curved symbols. *EuroTEX 2003 Proceedings*, *TUGboat* 24(3):323–327, 2003.

[11] Mostafa Banouni, Mohamed Elyaakoubi, and Azzeddine Lazrek. Dynamic Arabic mathematical fonts. *Preprints for the 2004 Annual Meeting*, Xanthi, Greece, pp. 48–53, 2004.

[12] Azzeddine Lazrek. RyDArab — Typesetting Arabic mathematical expressions. *TUGboat* 25(2):141–149, 2004.

[13] Thomas Milo. ALI-BABA and the 4.0 Unicode characters — Towards the ideal Arabic working environment. *EuroTEX 2003 Proceedings*, *TUGboat* 24(3):502–511, 2003.

[14] Yannis Haralambous and John Plaice. Multilingual Typesetting with Ω, a Case Study: Arabic. *Proceedings of the International Symposium on Multilingual Information Processing*, pp. 137–154, Tsukuba, 1997.

[15] George Williams. Font creation with FontForge. *EuroTEX 2003 Proceedings*, *TUGboat* 24(3):531–544, 2003; `http://fontforge.sourceforge.net`.