# FONT FORUM

Georgia K.M. Tobin

# A Handy Little Font

This short communication presents the code for a couple of dingbats that the author has found useful in memos and other correspondence. The code is quite straightforward, and can easily be put to use by the reader on any METAFONT implementation. Despite the simplicity of the code, there are a couple of interesting things done which I will enlarge upon a bit when we get to them.

The first part of the code looks like this:

```
% hands.mf
mode_setup;
size=48pt#;
font_size size;
em#:=size; cap#:=7/10em#; desc#:=3/10em#;
thinline#:=1/100em#;
define_pixels(em,cap,desc);
define_blacker_pixels(thinline);
```

Here we assign values to the height, depth and width of the character box and define the single pen size to be used. Since this is a very simple font, there is no call for overshoots or multiple pens; and the height and depth of the character box is just expressed as a fraction of the width.

Now, we define the whole character in a macro:

```
%Hand pointing right
def handpointing=
% define points for thumb and cuff
x1=x3=1/2[0,1/15w];
x2=x5=x4=x23=4/16w;
y1=y2=10/15[-desc,cap];
y3=y4=2/15[-desc,cap];
y5=6/7[y4,y2]; y23=1/7[y4,y2];
x6=9.75/16w;
y6=y2;
x7=11.25/16w;
y7=4/5[y23,y5];
x8=8.75/16w;
y8=1/4[y7,y6];
x17=14.5/16w;
y17=9.25/15[-desc,cap];
% find a point at a certain height on
% the curve from z6 to z7
 path dummyCurve; path dummyLine;
 x.dummy=1/2[x8,x7];
 y.dummy=y17;
 dummyCurve:=z6z5-z2..z7..tension1.4..z8;
 dummyLine:=z.dummy--z17;
 z18=dummyCurve intersectionpoint dummyLine;
x16=x17;
y16=y7;
x9=7/16w;
y9=y8;
x10=6/16w;
y10=2/5[y23,y5];
% find another point on the
% curve from z6 to z7
 x.dummy2=x5;
 y.dummy2=y16;
 x.dummy3:=1/2[x8,x7];
 y.dummy3=y.dummy2;
 dummyLine:=z.dummy3--z.dummy2;
 z12=dummyCurve intersectionpoint dummyLine;
% define points for curled fingers
x15=x14=x19=x22=1/3[x18,x17];
x13=x20=x21=x12;
y15=y16;
y13=y14=y15-(y17-y16);
y20=y19=y13-(y17-y16);
y21=y22=y20-(y17-y16);
% pick up pen and draw whole image
pickup pencircle scaled thinline;
draw z1--z2--z4--z3--cycle;
draw z5(1,1)..tension 1.5..z6
  &z6z5-z2...z7..tension 1.4..z8
```

```
  &z8down..tension3..z9
  &z9..tension 1.8..leftz10;
draw z18--z17right..z16--z7;
draw z7--z15right..z14--z13left..z12;
draw z14right..z19--z20left..z13;
draw z19right..z22--z21left..z20;
draw z21(-1,-1)..tension1.5..z23;
enddef;
```

The interesting bits I referred to at the start of this article are the lines in which *z18* and *z12* are defined. *z18* is the point at which the top of the pointing finger touches the top of the thumb; *z12*, the point at which the top of the second finger touches the underside of the thumb. Notice how METAFONT calculates the precise point of intersection easily, and obviates the need for fudging by the user.

Using this macro, we can write the code for a character depicting a hand pointing right in a few short lines; these lines, in fact:

```
beginchar("A",16/15em#,cap#,desc#);
handpointing;
endchar;
```

When we run this (and recall, *size* is set equal to 48 points), we get:

It would be nice to have a dingbat of a hand identical to the above except pointing left. We could describe such a character fairly easily by rewriting the code with all horizontal coordinates shifted; for example, the line:

```
x1=x3=1/2[0,1/15w];
```

would be rewritten:

```
x1=x3=1/2[w,14/15w];
```

But, there is a much simpler solution. We can write a file that redefines *endchar* in the same way that I discussed in "The ABC's of Special Effects" (*TUGboat,* Vol. 9, No. 1, April 1988, pp. 15-18):

```
% pattern_mirror
def pattern=
def endchar=
tracingequations:=1;
cullit;
picture normalchar;
normalchar:=currentpicture;
picture mirrorimage;
  mirrorimage:=normalchar
  reflectedabout ((0,0),(0,h))
  shifted (w,0);
currentpicture:=mirrorimage;
scantokens extra_endchar;
chardx:=w;
shipit;
if displaying>0:  showit; fi
endgroup;
enddef;
enddef;
```

This redefinition will, in essence, capture the image shown above, reflect it about the y-axis, and shift it to the right by the character width. We can input that file within another character definition, call *pattern*, and then call our *handpointing* routine:

```
beginchar("B",16/15em#,cap#,desc#);
input pattern_mirror;
pattern;
handpointing;
endchar;
```

Running the above (and recall, size is set equal to 48 point) we get:

We can make use of another file providing another definition of *pattern*, namely:

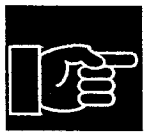
```
% pattern_rev
def pattern=
def endchar=
tracingequations:=1;
cullit;
picture phaseone; phaseone=currentpicture;
currentpicture:=nullpicture;
fill (0,-desc)--(w,-desc)--(w,cap)--
     (0,cap)--cycle;
cullit;
picture phasetwo;
phasetwo=currentpicture-phaseone;
currentpicture:=phasetwo;
scantokens extra_endchar;
chardx:=w;
shipit;
if displaying>0:  showit; fi
endgroup;
enddef;
```

```
enddef;
```

This redefinition captures the image drawn above and removes each pixel in it from an entirely blackened character box. As before, we input that file within another character definition, call *pattern* and then call the *handpointing* routine; thus:

```
beginchar("C",16/15em#,cap#,desc#);
input pattern_rev;
pattern;
handpointing;
endchar;
```

and get:



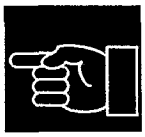
Finally, we can use another file that combines the features of the previous two patterns:

```
% pattern_revmirror.mf
def pattern=
def endchar=
tracingequations:=1;
cullit;
picture phaseone; phaseone=currentpicture;
currentpicture:=nullpicture;
fill (0,-desc)--(w,-desc)--(w,cap)--
     (0,cap)--cycle;
cullit;
picture phasetwo;
phasetwo=currentpicture-phaseone;
picture phasethree;
phasethree:=phasetwo
  reflectedabout ((0,0),(0,h))
  shifted (w,0);
currentpicture:=phasethree;
scantokens extra_endchar;
chardx:=w;
shipit;
if displaying>0:  showit; fi
endgroup;
enddef;
enddef;
```

and which, accessed by this code:

```
beginchar("D",16/15em#,cap#,desc#);
input pattern_revmirror;
pattern;
handpointing;
endchar;
```

gives us the reverse video mirror image of the original character:



The user can, of course, change the second line of the code shown above to create the font at any size desired; you might wish to actually put that line outside the rest of the code, and simply specify it at run time.

To create the font, all you need is the code shown in this article and METAFONT. Start up METAFONT, specify the device for which you are creating the font by typing `mode=lowres;` (or `mode=` whatever you wish) and then inputting *hands.mf*.