



# Another Dingbat Idea

I take pen in hand to describe the design and coding of a simple dingbat. I hope that this will inspire all you would-be METAFONTers to try your hands, heads and keyboards at creating entries for the Dingbat Competition, announced in the last issue of *TUGboat*.

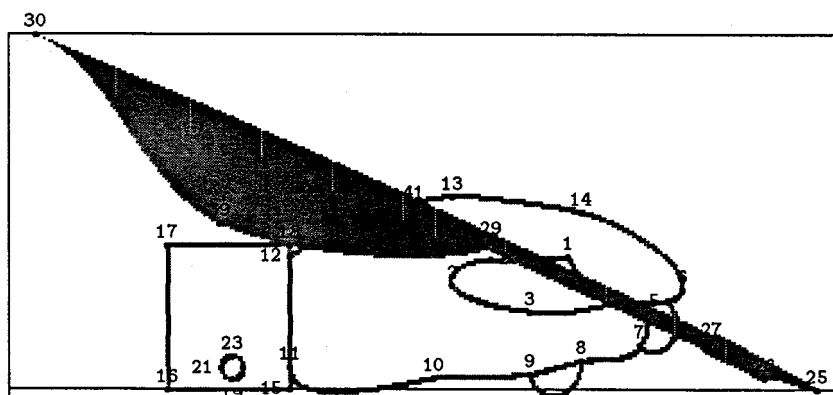
I started with a sketch of a left hand holding a quill pen (left, because I had to draw with my right) and marked what I figured would be the necessary points to describe the figure with METAFONT. Of necessity, the figure is simplistic, somewhat cartoonish; remember that all the details that you lovingly describe on your initial large sketch (mine was approximately 180 points high and 480 points wide) must survive down to 10 points!

The next step was to lay out a grid and orient the figure on it relative to the baseline and width. Since I do not want any of the character to lie under the line of text next to which it occurs, I oriented the bottom of the hand and the cuff exactly on the baseline. However, I did add a wee bit of descender for the reverse video case. And there's no need for

any tricky machinations in terms of part of the figure lying to the left of the line  $x=0$  or to the right of  $x=\text{charwidth}$ ; the apparent width of the character is the real width we want  $\text{T}\text{E}\text{X}$  to know about.

My next step was to divide the total width and height into some reasonable number of units. One caveat here: don't make the grid too fine, or you'll tend to overdo the number of points you select as key points and over-analyze the character. Think of the design process as a collaborative effort with METAFONT, rather than as an attempt to control it. As Don Knuth has said, some of the most fruitful parts of design can occur when you let METAFONT "have its own head".

Now, I selected my key points. I cannot stress too much the need to be a little freewheeling from now on. The fact that I believed a point was key at this early stage of the design process ought not to force me to keep that point in later on. In a bit, you'll see the code for the figure, where I have left the numbering of the points as I originally did them to illustrate this. You'll notice some gaps between  $z30$  and  $z41$ .



$$22 = 19 + (4.3, 4.3);$$

$$28 = 4 + (0, 0);$$

FIGURE 1: Proofmode drawing of character.

My original sketch involved a much fancier feather on the quill, which just didn't work at a design size of 10 points. I opted for a plainer feather, and removed a number of points. Figure 1 shows a proof mode of the characters as finally produced, with the numbering of the points as shown in the code.

I was now ready to start writing stuff that resembles METAFONT code; and this is it:

```
% define points for hand
x1=10.75/16w; y1=3/8h;
x2=8.5/16w; y2=2.35/8h;
x3=x9=10/16w;
y3=1.75/8h; y9=0.35/8h;
x4=11.75/16w; y4=2/8h;
x5=12.65/16w; y5=2/8h;
x6=12.95/16w; y6=2.25/8h;
x7=12.125/16w; y7=1/8h;
x8=11/16w; y8=0.65/8h;
x10=8.125/16w; y10=0.25/8h;
x20=9.25/16w; y20=2.95/8h;
x13=8.5/16w; y13=4.35/8h;
x14=11/16w; y14=4/8h;
% two points on the wrist that touch
% the cuff, and the cuff
x11=x12=x15=x18=5.35/16w;
y11=0.5/8h; y12=3/8h;
y15=0; y18=3.25/8h;
x16=x17=3/16w; y16=y15; y17=y18;
% define the button
x19=x23=good.x 4.25/16w;
y19=good.y 0.25/8h;
y23=y19+0.5/8h;
x19=1/2[x21,x22];
x21=x22-(y23-y19);
y21=y22=1/2[y19,y23];
x25=15.5/16w; y25=0;
x26=14.5/16w; y26=0.25/8h;
penpos27(quillWidth,50);
penpos28(quillWidth,50);
penpos29(quillWidth,50);
penpos41(quillWidth,50);
z28=z4; x29=x20;
x27=13.5/16w;
z29=whatever[z25,z28];
z27=whatever[z25,z28];
x30=0.5/16w; y30=h;
x39=4/16w; y39=3.75/8h;
x41=7.75/16w; z41=whatever[z29,z27];
```

You'll guess that my grid was 8 units high and 16 units wide. You'll note, too, that the leftmost point

is just a bit greater than 0, and the right a bit less than  $w$ ; this will account for sidebearings at either side of the character. Most all these points are stated in terms of the grid, rather than in terms of relation to one another; but remember to use such relationships whenever they are pertinent to the design. For example, the last two lines above define where  $x41$  and  $y41$  lie; but what is important is not the precise location of  $y41$  on the grid, but the fact that the point lies somewhere on the line between  $z29$  and  $z27$ . Needless to say, don't be shy about articulating the precise nature of these relationships in comments in your code.

Once I had established the location of all the key points on the character, I was reminded of one of my favorite Monty Python sketches: a Shakespearean actor elucidates on his craft thus: "It's not just a question of the number of words. You have to get them all in the right order." This is pretty much the next step in our design process: I have established a reasonable number of points, and now have to get them all in the right order, by writing the code to connect them in pleasing ways. This is the code I came up with:

```
% draw the hand
pickup pencircle scaled penWidth;
draw z1---z20{left}..z2..tension1.6..z3..z4
    &z4{left}..tension 1.6..z1;
draw z12..tension 1.6..z13..
    tension 1.8..z14..
    tension 1.6..z6..z5..{left}z4;
draw z4{right}..z7..z8..tension 1.3..z9..
    tension 1.4..z10..tension 1.4..{up}z11;
draw z5{(1,-1)}..{(-1,1)}z7;
draw z8{down}..{up}z9;
% draw the cuff
draw z18--z17--z16--z15--cycle;
% draw the button
draw z19..z21..z23..z22..cycle;
% draw the quill
filldraw z27r--z25
    &z25{z28r-z27r}
    ..tension 2..{(-1,-1)}z26
    &z26..{z28l-z27l}z27l
    &z27l--cycle;
penstroke z29e--z27e;
fill z29r---z41r..z30
    &z30{z29-z30}..z39..tension1.4..z29l
    &z29l--cycle;
```

Niceties like amount of tension between points or direction desired entering or leaving points are (at

least in my experience) only rarely coded correctly first time out. This is where you and METAFONT get to work closely together. Draft some code, see what METAFONT does with it, and then tune on the basis of what you see. Often, you'll be pleasantly surprised with improvements that sneak into the design as you work.

But, of course, the code above is not yet ready for a collaborative effort with METAFONT. We have to attend to some housekeeping first. If you start the lines above with

```
beginchar("A",w#,cap#,desc#);
```

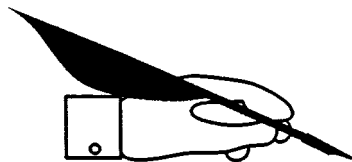
and conclude them with `endchar;` and preface them with some code that specifies font size, width, pens and so on:

```
%% Pen and character box info.
%% Set mode for device to print on
\mode=localfont;
mode_setup;
font_size 60pt#;
em#:=60pt#; cap#:=.95em#;
desc#:=.025em#; w#:=16/7em#;
overshoot#:=.025em#;
penWidth#:=em#/60; quillWidth#:=em#/20;
define_pixels(em,cap,desc,w,overshoot);
define_blacker_pixels(penWidth);
define_blacker_pixels(quillWidth);
```

and since you'll probably at some point want to see a proofmode character printed out with all the points numbered, include the lines:

```
labels(1,2,3,4,5,6,7,8,9,10,11,12,13,
       14,15,16,17,18,19,20,21,22,23,
       24,25,26,27,28,29,30,39,41);
```

However, with the mode set above, you won't get a proofmode character, but a character suitable for printing on your local device, namely:



So far, so good. But, I knew I wanted a right hand version as well (in fact, *all* I really wanted was the right hand version!) I did not even briefly consider recalculating the positions of all the points to flip the character. I could have simply copied all the code for the character above, given it a new code number and concluded it with a `rotatedabout;` but it seemed

much tidier to make the code for the dingbat proper a macro. So, start the lines of code describing the dingbat not with `beginchar` but with

```
def HandWithQuill=
```

and conclude them, not with an `endchar` but with

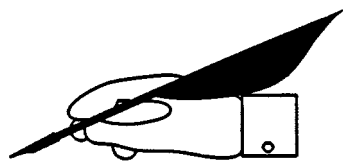
```
enddef
```

The code for the character with code "A" shown above is then condensed to

```
beginchar("A",w#,cap#,desc#);
HandWithQuill;
endchar;
```

and a mirror image version (the sought-after right hand dingbat) is coded as

```
beginchar("B",w#,cap#,desc#);
HandWithQuill;
currentpicture:=currentpicture
  reflectedabout ((0,0),(0,h))
  shifted (w,0);
endchar;
```



Since it is so easy to manipulate the `currentpicture`, we might as well produce a reverse video of the original left hand dingbat, with this code:

```
beginchar("C",w#,cap#,desc#);
HandWithQuill;
cullit;
picture savecurrent;
  savecurrent=currentpicture;
clearit;
% blacken the whole character box
fill (0,-desc)--(w,-desc)--
  (w,cap+overshoot)--(0,cap+overshoot)
  --cycle;
currentpicture:=
  currentpicture-savecurrent;
endchar;
```

(Note well the `cullit` and `clearit`!) and produce




and code a right-handed version like so:

```
beginchar("D",w#,cap#,desc#);
HandWithQuill;
% Flip the image first
currentpicture:=currentpicture
  reflectedabout ((0,0),(0,h))
  shifted (w,0);
cullit;
% Save the flipped image
picture savecurrent;
  savecurrent=currentpicture;
clearit;
% Blacken the character box
fill (0,-desc)--(w,-desc)--
  (w,cap+overshoot)--(0,cap+overshoot)
  --cycle;
currentpicture:=currentpicture-savecurrent;
```

endchar;

for the result



I hope that this description of the design and coding of a simple dingbat will encourage the reader to attempt one, or better yet, several. While the design of a complete font with METAFONT is a difficult and sometimes tedious process, the creation of a simple dingbat and some handy variations on it is not. It provides an enjoyable introduction to the use of METAFONT, and might just produce a dingbat that the reader can use to enhance and to personalize T<sub>E</sub>Xed documents. 

---

ERRATUM: "A Handy Little Font", Font Forum, *TUGboat*, Volume 10, No. 1

I regret that, inadvertently, I did something underhanded in my last *Font Forum* – to wit, I neglected to make the left and right braces visible in the code. My apologies to all who heeded the largish admonition at the end to "TRY IT", who were rewarded only with surly messages from METAFONT.

The macro for the whole handpointing character should read like this:

```
%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:=z6{z5-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
  &z6{z5-z2}...z7..tension 1.4..z8
  &z8{down}..tension3..z9
  &z9..tension 1.8..{left}z10;
draw z18--z17{right}..z16--z7;
draw z7--z15{right}..z14--z13{left}..z12;
draw z14{right}..z19--z20{left}..z13;
draw z19{right}..z22--z21{left}..z20;
draw z21{(-1,-1)}..tension1.5..z23;
enddef;
```