UTF8 in MetaPost

Hans Hagen

Around the time Alan Braslau and I were discussing his new MetaFun node module, I made the MetaPost library accept UTF8, if only because nowadays this is the preferred portable file encoding. Before I show you what that brings us, let's see how the TEX suite deals with input.

When TEX and MetaFont, the program that MetaPost shares much of its code with, evolved, punch cards were still widely used. There was quite some diversity in the word sizes of computers and these were not always in multiples of eight. Initially characters in the engines took seven bits but soon that became eight. When a character was read from file, it went through a re-mapper that turned the input byte into one that was normalized inside the engine. Before something was shown to the user (on the console or in the log), there was a conversion to the preferred output encoding. Internally ASCII was used, but the outside world could, for instance, talk EBCDIC.

In the previous paragraph I talk in the past tense because in the extended TEX engines that I use, LuaTEX and LuaMetaTEX, this mapping no longer exists: they have a UTF8 code path instead. In the MetaPost library that is used in LuaMetaTEX, the mapping has also disappeared because in practice it was a one-to-one mapping, an unused leftover from the past (one can consider it an old system dependency).

The TEX and MetaPost engines differ in the way that they deal with characters. In TEX a character has a so-called catcode. For instance, a dollar sign is a math shift character (it begins or ends math mode) and its code is 3. A space has code 10, a comment 14, etc. There are 16 catcodes and if you want to know more about them: read the TEX book! It's one of these intriguing properties of TEX.

In MetaPost characters don't have catcodes but they are grouped into classes that drive the expression scanner, like left or right bracket. They also play a role in prioritizing operators. In TEX, characters with a code larger than 127 are valid, and depending on how a macro package is set up, they have category 'letter' or 'other'. In stock MetaPost they are illegal. However, in MetaPost we can make them letters (one of the classes) after which the engine will just accept them and not complain. In the wide TEX engines, the characters larger than 127 signal a multibyte UTF8 sequence, which also means that the related character code ends up as glyph reference. If you want a specific UTF sequence to be a valid letter in a macro name, you need to make sure it has the right catcode: you need to set this up (think of Chinese with thousands of characters). In MetaPost it's easier: just put all the characters of the 128-255 range in the 'letter' class and you're done. You can tell the library to do that with a simple flag, and MetaPost can do UTF8. All it takes is this:

```
for (int k = 127; k <= 255; k++) {
    mp->char_class[k] = mp->utf8_mode
                         ? letter_class
                         : invalid_class;
}
```

After this rather trivial patch (of course one needs to set the mode) we can do the following to get figure 1:

```
\startMPcode
vardef dœn_knüth   = textext("Don Knuth")    enddef ;
vardef ДональдКнут = textext("Donald Knuth") enddef ;

draw             ДональдКнут  xsized 10cm             withcolor "middlegray";
draw             dœn_knüth    xsized  4cm             withcolor "darkred";
draw             dœn_knüth    ysized  5mm rotated  45 withcolor "darkgreen";
draw textext(str dœn_knüth)   ysized  5mm rotated -45 withcolor "darkblue";
draw textext(str ДональдКнут) ysized  5mm rotated  90 withcolor "darkgray";
\stopMPcode
```

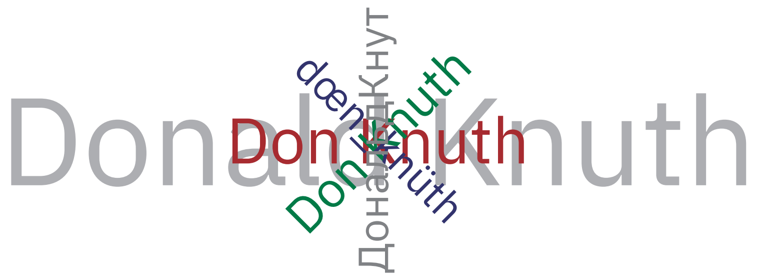

Figure 1.

But, as Alan and I were playing with this, a more tantalizing example became possible; the result is shown in figures 2 and 3:

```
\startMPcode
  save p ; pen p ; p := currentpen ;
  pickup pencircle scaled .05;

  picture ○ ; ○ := image (draw fullcircle) ;
  picture ◎ ; ◎ := image (draw fullcircle ; draw fullcircle scaled .5) ;

  currentpen := p ;

  draw ◎ ysized 2cm withcolor "darkblue" ;
  draw ○ ysized 2cm shifted (4cm,0) withcolor "darkred" ;
\stopMPcode
```

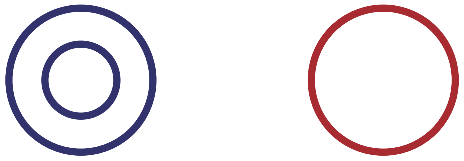

Figure 2.

```
\startMPcode
  draw image (
    for i=1 upto 100:
      draw ⊚ scaled .3i shifted ((i/2)*mm,0) rotated (i*10)
             withcolor (i*red/100);
    endfor ;
  ) shifted (4cm,8cm);
\stopMPcode
```

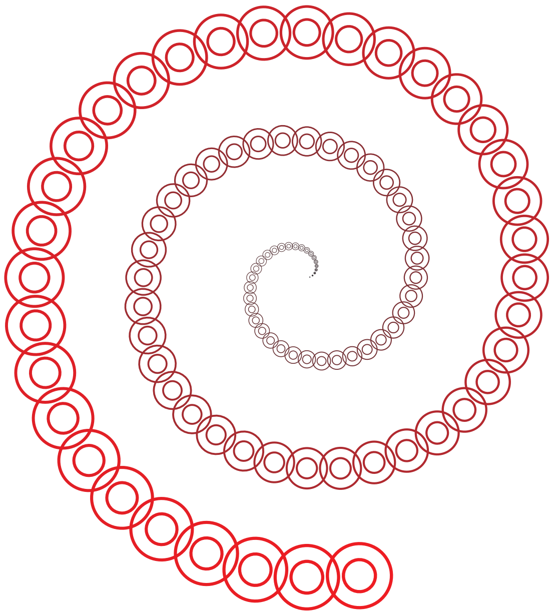

Figure 3.

In the end we came up with a bunch of symbols that can be used as indicators in graphics for tagging data points:

```
\startMPcalculation
begingroup
  pen savedpen ; savedpen := currentpen ;
  pickup pencircle scaled .05 ;
  interim ahlength := .5 ;
  interim ahvariant := 1 ;
  picture ○ ; ○ = image(draw fullcircle) ;
  picture ⊚ ; ⊚ = image (draw fullcircle ;draw fullcircle scaled .5) ;
  picture □ ; □ = image(draw fullsquare) ;
  picture ◇ ; ◇ = □ rotated 45 ;
  picture △ ; △ = image(draw (dir 90--dir 210--dir 330--cycle) scaled (2/3)) ;
  picture ▽ ; ▽ = △ rotated 180 ;
  picture ◁ ; ◁ = △ rotated  90 ;
  picture ▷ ; ▷ = △ rotated -90 ;
  picture ● ; ● = image(fill pathpart ○) ;
  picture ■ ; ■ = image(fill pathpart □) ;
  picture ◆ ; ◆ = image(fill pathpart ◇) ;
  picture ▲ ; ▲ = image(fill pathpart △) ;
  picture ▼ ; ▼ = image(fill pathpart ▽) ;
  picture ◀ ; ◀ = image(fill pathpart ◁) ;
  picture ▶ ; ▶ = image(fill pathpart ▷) ;
  picture ↑ ; ↑ = image(drawarrow (0,-1/2)--(0,1/2)) ;
              setbounds ↑ to unitsquare;
  picture → ; → = ↑ rotated 180;
  picture ↓ ; ↓ = ↑ rotated  90;
  picture ← ; ← = ↑ rotated -90;
  pickup savedpen ;
endgroup ;
\stopMPcalculation
```

These symbols can now be used as follows, see figure 4 for the result:

```
\startMPcode
save n ; n := 0 ;
for symbol = ○, ◎, □, ◇, △, ▽, ◁, ▷, ●, ■, ◆, ▲, ▼, ◀, ▶, ↑, →, ↓, ← :
  draw symbol scaled 4mm shifted (n, 0) ;
  n := n + 6mm ;
endfor ;
\stopMPcode
```

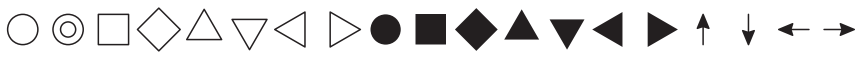

Figure 4.

Of course, when we have many such symbols, using a font with these characters in combination with the `textext` command is more efficient because then we just refer to a shape in a font.

The possibilities are endless. Take the following:

```
\startMPcalculation
  def ○ = fullcircle enddef ;
  def ✁ = cutafter enddef ;
  def ✃ = cutbefore enddef ;
  def ✏ = withpen pencircle enddef ;
  def ✖ = scaled enddef ;
  def ◷ = rotated - enddef ;
  def ◴ = rotated enddef ;
\stopMPcalculation
```

This might draw icon and emoji freaks to MetaPost, but it might equally drive away potential users (see figure 5):

```
\startMPcode
  draw (○ ✁ point 2 of ○) ✖ 2cm ✏ ✖ 5mm ◷ 90 withcolor "darkred" ;
  draw (○ ✃ point 2 of ○) ✖ 2cm ✏ ✖ 5mm ◴ 90 withcolor "darkblue" ;
\stopMPcode
```

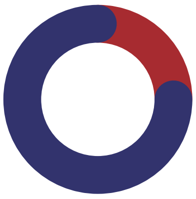

Figure 5.

But, as with many obscure features in macro packages, I'm sure that users will find a way to (ab)use this feature. Just for the record: the `textext` command is the MetaFun way to get typeset text, and using string for colors is just a convenient way to access colors at the TEX end (a redefinition of a primitive). The `MP` wrapper commands deal with runtime MetaPost processing and embedding.

This article was first published in NTG's MAPS No.51.

Last touches to the statistical charts module   Photo: Harald König