Another Approach to Barcodes

Peter Willadt

Abstract

This article copes with barcodes, in particular with interleaved two-of-five and with code 39. It shows various means to generate them with TEX and related software, even the use of VF-files for barcode generation.

1 Background

1.1 Some Common Barcode Types

In stores, upc and ean codes are widely used for automatic identification, pricing, etc. ean barcodes have already been covered in *TUGboat* [1].

In an industrial environment, other barcodes are in use. This article deals especially with two of them, interleaved two-of-five (ITF for short) and code 39.

Original two-of-five code consists of five bars: two of them thick and three thin. With the ten different possible combinations, just the ten digits could be coded. Original two-of-five ignores the width of the gaps between the bars and so it wastes lots of space. One remedy against that was the creation of barcodes where the gaps became meaningful. With ITF, however, there was the somewhat unlucky decision to use the gaps not in a way where one digit would be coded with three bars and two gaps, but to code two digits together — the first digit within the bars and the second one within the gaps (that's where the *interleaved* comes from).

As a consequence, ITF is quite easy to produce by a program, but it is very hard to generate with a text processor or within a report generated from database output. Also you can only code an even number of digits. With quantities, this is no problem, but with fixed-length article codes, things may be different.

The us postnet code is a special kind of two-of-five, where the bars have different height instead of different width. It has already been covered in-depth in *TUGboat* [3, 2].

Code 39 consists of 9 elements — five bars and four gaps. Three of these nine elements are thick, the others are thin (hence *three-of-nine*). As this gives a lot of possible combinations, code 39 can be used not only for digits, but also for uppercase letters from A to Z and seven special signs (+−/ dot, space, dollar, and percent). The bad news about code 39 is that it also takes lots of space. Coding an eight-digit number in medium resolution may easily take two inches of space.

Figure 1: Codabar looks like this. If you have a barcode reader, you should read c1009*

Codabar consists of seven elements — four bars and the three gaps between them. The meaning is contained in two or three thick elements, hence codabar could also be called two-or-three-of-seven. Codabar can be used for the ten digits and + − / :, dot, and dollar. Strangely enough, codabar uses four sets of start/stop-signs. You have to use either the set a/t, b/n, c/* or d/e. The start/stop-signs will be decoded together with the digits between them.

1.2 Usage with TEX

TEX sometimes is used as a back end for database output. I use TEX also for the creation of labels for pharmaceutics for in-house use.

With the ligature mechanism of TEX, ITF output can be easily and transparently handled. And, through TEXs ability to draw bars, also the creation of barcodes without using any other software is possible.

1.3 Readability

Barcodes look rather ugly, so one tends to make them as small as possible. But for readability, there are limits. For a normal scanning device, the width of a narrow bar in code 39 or in ITF must not be less than $7.5 \cdot 10^{-3}$ inches (≈ 0.2 mm). In ITF and code 39, thick gaps or bars have to be at least twice as thick as thin gaps or bars. In higher resolutions, making them 2.25 to 3 times as wide may yield better results.

The bleeding of the bars has not only to be considered for the output device of TEX, but if there is a printing step to follow, the bleeding of the secondary printer has to be compensated, too — at least at higher bar code resolutions. Code 39 and ITF should appear to be about 50% black, not much darker. The contrast between barcode and background should be high. As most reading devices use red light, the bars should not be printed red.

To the left and right of the barcode, there has to be some space in background color. The higher the bars are, the more the reader can be rotated with respect to the bars. If the bars are as high as the complete code is wide, the reader can be twisted at an angle of ±45 degrees.

It is always best to have access to a reading device to verify good readability.

2 Making Barcodes Work

There are different approaches that fortunately all work. The first approach is to use TEX to draw the barcodes entirely on its own. The great advantage is complete portablity and also an independent control over both height and width of the barcodes. The disadvantage is that macros have to be used for every bar that is to be shown. This may lead to some problems and also to slow run-time behaviour. When coding ITF, the macros get rather complicated.

Another approach would be to use METAFONT to draw the barcode characters. This approach is straight forward. Also with METAFONT there is very fine control over bleeding edges, etc. This control is especially useful with high-density barcodes or with low-quality printers. The disadvantage of using METAFONT is the requirement to create the barcodes at all needed resolutions.

The third approach uses a virtual font (VF-) file to do all the drawing. As a virtual font can be used not only to map from one font or charset to another, but also to draw bars, we get a flexible solution. To TEX the barcodes are just characters from another font; the DVI driver just has to draw rules. The main disadvantage is that we lose the fine control over printer-dependent bleeding, etc., so that I recommend using this solution only for low- to medium-resolution barcodes. Another disadvantage lies in the absolute lack of meta-ness: in VPL-files, everything has to be coded by hand and spelled out explicitly. Any change required results in a lengthy editor session.

2.1 Barcodes by TEX

It is not difficult to draw bars. But drawing barcodes requires drawing a lot of bars. Having defined macros for the obvious start/stop-code and the individual characters, one can draw barcodes by invoking the control sequences directly or — more elegantly — by making characters active and letting them draw themselves. The third solution — reading the characters to draw as parameters to a macro — works well for a numerical-only barcode, but 43-way branching as would be necessary for code 39 is surely not attractive. Here is a short excerpt from the coding of code 39.

```
\newdimen\dick     \newdimen\dickbar
\newdimen\duenn    \newdimen\duennbar
\newdimen\antibleed
\dick=1.2mm        \duenn=0.6mm
\antibleed=0mm
\dickbar=\dick     \duennbar=\duenn
```

Figure 2: Code 39 looks like this. These lines mean *HELLO*

```
\advance\dickbar by -\antibleed
\advance\duennbar by -\antibleed
   % b means bar...
\def\b{\vrule width\duennbar}
\def\B{\vrule width\dickbar}
   % ...and s means space
\def\s{\hskip\duenn\hskip\antibleed}
\def\S{\hskip\dick\hskip\antibleed}
   % and then the coding
\def\tninestart{\b\S\b\s\B\s\B\s\b\s}
\def\tnineminus{\b\S\b\s\b\s\B\s\B\s}
\def\tninelettera{\B\s\b\s\b\S\b\s\B\s}
\def\tnineleletterb{\b\s\B\s\b\S\b\s\B\s}
   % many lines of code omitted
\def\makethemactive{%
    \catcode`\A=\active
    \catcode`\B=\active
   % many lines of code omitted
}
{\makethemactive
 \gdef\begincodethirtynine{%
    \bgroup\makethemactive
    \strut\tninestart
    \let A=\tninelettera
    \let B=\tnineletterb
   % many lines of code omitted
 } % end begincodethirtynine
}  % end scope makethemactive
\def\endcodethirtynine{%
    \tninestart\egroup
}
```

The text to be printed as a barcode can then be included between `\begincodethirtynine` and `\endcodethirtynine`.

The code width and height can easily be adjusted from TEX, and the extra blackness added by the printing engine can also be removed by adjusting `\antibleed`. The coding is somewhat fuzzy, because many characters, even the space, have to get a special meaning already where `\begincodethirtynine` is being defined.

If only digits have to be printed, a multiway branch may be suitable to draw the bars and the coding can be handled by tail recursion. The following example shows the coding of a *Pharmazentralnummer*. These are article numbers for german pharmaceuticals. They consist of exactly seven digits; in code 39, they are preceded by a minus sign. The last digit is a weighted mod 11 checksum. In the following example, besides drawing bars, TEX also counts the digits and calculates the checksum.

```
% needs the code printed above
% three counts for checksumming etc.
\newcount\ziffern
\newcount\checksum
\newcount\multreg
% numbers in code 39
 \def\tnzero{\b\s\b\S\B\s\B\s\b\s}
  \def\tnone{\B\s\b\S\b\s\b\s\B\s}
  \def\tntwo{\b\s\B\S\b\s\b\s\B\s}
\def\tnthree{\B\s\B\S\b\s\b\s\b\s}
 \def\tnfour{\b\s\b\S\b\s\b\s\B\s}
 \def\tnfive{\B\s\b\S\B\s\b\s\b\s}
  \def\tnsix{\b\s\B\S\B\s\b\s\b\s}
\def\tnseven{\b\s\b\S\b\s\B\s\B\s}
\def\tneight{\B\s\b\S\b\s\B\s\b\s}
 \def\tnnine{\b\s\B\S\b\s\B\s\b\s}
\def\tndigit#1{%
  \ifcase#1\tnzero\or\tnone
    \or\tntwo\or\tnthree
    \or\tnfour\or\tnfive
    \or\tnsix\or\tnseven
    \or\tneight\or\tnnine
    \fi%
}
\def\endtncode{%
  \ifnum\ziffern=9
    \else\message{wrong digits count}%
  \fi
  \ifnum0=\checksum
    \else\message{wrong checksum}%
  \fi
  \tninestart\egroup
}
\def\nexttn#1{%
  \advance\ziffern by1
  \if@#1\let\next\endtncode
   \else
   \tndigit#1
% begin checksum stuff
   \ifnum\ziffern=8
    \multreg=\checksum
    \divide\multreg by 11
    \multiply\multreg by 11
    \advance\checksum by-\multreg
    \multreg=#1
    \advance\checksum by-\multreg
    \ifnum\checksum=10
        \checksum=0
    \fi
   \else
    \multreg=#1
```

```
    \multiply\multreg by\ziffern
    \advance\checksum by\multreg
   \fi
  \fi
% end checksum stuff
  \next
}
\def\pzncode{%
  \bgroup
  \let\next\nexttn
  \ziffern=1\checksum=0\multreg=0
  \strut
  \tninestart\tnineminus%
  \next
}
% Example (right)
\pzncode1234562@
% Example (wrong)
\pzncode1235462@
```

2.2 Barcodes by METAFONT

The advantages of METAFONT are quite clear: It is easy to draw bars, to build characters out of them, to cope with the printer's bleeding and similiar effects (the german word is *Druckzuwachs*), and, because of the meta-ness, very compact and flexible code can be written. When the bars have to be combined with other elements (ean code, e.g., requires the number printed in ocr under the bars) METAFONT is the first choice.

Using pk fonts, the fonts produced are rather compact — they may be even shorter than the corresponding VF-files.

As this article already tends to be lengthy, I will not treat METAFONT any further.

2.3 Barcoding with virtual fonts only

Virtual fonts can contain any instruction that is also found in a DVI file. So, besides typesetting letters and moving, virtual fonts can also contain instructions to draw bars. As an example, I have completely mapped ITF barcodes into a virtual font. The handling of two digits sharing the same bars and gaps is left entirely to the ligature mechanism, so the font contains a little more than a hundred characters.

The header of the VPL file looks like the header of any VPL file for a monospaced, upright font.

```
(FAMILY BARCODE)
(DESIGNSIZE D 12)
(DESIGNUNITS D 14)
(COMMENT written by Peter Willadt)
(COMMENT August 16, 1997)
(FONTDIMEN
   (SLANT   R  0)
   (SPACE   D 14)
   (SHRINK  D  0)
   (STRETCH D  0)
   (XHEIGHT R 10)
   (QUAD    D 14)
   )
```

Everything quoted can be found in the documentation for VPtoVF, so I will not discuss it in detail. Just let me mention that the design size of the font is twelve points, and that these twelve points are divided into fourteen units (five bars and five gaps, and two thick bars and gaps each). Thin units will be about 0.3 mm; that makes a medium resolution.

After the heading, there follows an extensive `ligtable`, where every combination of each digit with any other digit is mapped to a character. Here is a short excerpt, showing the zero preceding any digit:

```
(LIGTABLE
   (LABEL C 0)
   (LIG   C 0 D 1) (LIG   C 1 D  2)
   (LIG   C 2 D 3) (LIG   C 3 D  4)
   (LIG   C 4 D 5) (LIG   C 5 D  6)
   (LIG   C 6 D 7) (LIG   C 7 D  8)
   (LIG   C 8 D 9) (LIG   C 9 D 10)
   (STOP)
```

The rest is too dull to show here. The characters from 48 to 58 have not been used for ligatures; this not only gives clear code, but also avoids reentering the ligature mechanism.

Start- and stop-codes are different for ITF. I have mapped them to the + and − keys. Codings for 0–9 themselves are contained in the font, because there is no possibility to have ligatures when the constituent characters do not exist. They only get drawn when there is an odd number of digits to code; this is a case that should never happen. So there are two possibilities: make the mistake immediately visible (e.g. by just skipping instead of drawing) or make the best of it. I chose the latter way and mapped these codes to the same chars that would result from the digit drawn twice — hoping that an excess digit at the end would do no harm.

Let us just have a short look at the character that will be drawn when a 0 is followed by a 5:

```
(CHARACTER D 6
   (COMMENT 0 and 5)
   (CHARWD D 14)     (CHARHT R 14)
   (CHARDP R  0)     (CHARIC R  0.0)
   (MAP
      (SETRULE R 14 R 1)(MOVERIGHT R 2)
```

Hello, the plate with the mix computer's serial number looks like this:

Figure 3: ITF code. Output of the sample code.

```
    (SETRULE R 14 R 1)(MOVERIGHT R 1)
    (SETRULE R 14 R 2)(MOVERIGHT R 2)
    (SETRULE R 14 R 2)(MOVERIGHT R 1)
    (SETRULE R 14 R 1)(MOVERIGHT R 1)
    )
)
```

All the action consists of drawing several bars, each 14 units (12 pt) high, and 1 or 2 units wide, and skipping 1 or 2 units after every bar.

VPL files are human-readable, but I guess they were not intended to be human-writable. You have to painfully avoid using the tab-key for indenting; also, it is recommended to indent in a very intentional way, lined up as shown in the example, and with every level of indentation shifted by the same number of spaces further to the right.

Having the VPL file, you simply have to make TFM and VF files out of it. You do this by invoking VPtoVF and copying the resulting files to locations where TEX and the printer driver can find them.

Within TEX, ITF can then simply be used like any other font; it looks like this:

```
\font\itf=wlitf scaled 2000
\def\itfcode#1{{\itf+#1-}}
Hello, the plate with the
mix computer's serial number
looks like this:
\centerline{\itfcode{1009}}
```

Higher TEXnique would require checking whether the argument to `\itfcode` is completely numeric and if it consists of an even number of digits — maybe even calculating a checksum. But for processing database output, the checking should happen at an earlier stage.

Because of the possibility to include references to other fonts within a VPL file, it is also possible to build a virtual font containing bars and digits — something that you need for upc codes etc. — provided you have already got a font of ocr digits.

3 Conclusion

You can create barcodes any way you like; TEX offers several opportunities. The conventional method of using METAFONT is surely the best way, but for one-time-use TEX itself may suffice. Using a VPL-file alone results in scalable fonts, but without hinting.

The files related to this article (code 39 both in METAFONT and TEX macros, an ITF font and a codabar font, both in VPL format, and an ean font in METAFONT format) can be found on CTANin `../tex-archive/fonts/barcodes/willadt/`. You should be aware of the fact that for the METAFONT code, you need to do some preprocessing. All fonts and macros may be freely used without restrictions.

References

[1] Peter Olšak. The EAN barcodes by TEX. *TUGboat*, 15(4):459–464, 1994.

[2] John Sauter. Postnet codes using METAFONT. *TUGboat*, 13(4):472–476, 1992.

[3] Dimitri Vulis. TEX and envelopes. *TUGboat*, 12(2):279–284, 1991.

◇ Peter Willadt
Heinrich-Wieland-Allee 5
75177 Pforzheim
Germany
`Willadt@t-online.de`