Package pst-barcode
v. 0.03

Terry Burton[*] Herbert Voß[†]

October 24, 2005

file name	*meaning*	*version*
pst-barcode.sty	LaTeX style file – wrapper	2005-07-24
pst-barcode.tex	TeX file – PS interface	2005-07-26
pst-barcode.pro	PostScript file	2005-10-17
barcode.tex	documentation source	2005-09-16
barcode.pdf	documentation PDF	2005-09-16

[*]tez@terryburton.co.uk
[†]hvoss@tug.org

Contents

1 Introduction

To install the package put the three files in a place, where TEX will search for the files:

name	*target dir*
`pst-barcode.tex`	`$LOCALTEXMF/tex/generic/pstricks/`
`pst-barcode.sty`	`$LOCALTEXMF/tex/latex/pstricks/`
`pst-barcode.pro`	`$LOCALTEXMF/dvips/pstricks/`
`barcode.tex`	`$LOCALTEXMF/doc/pstricks/`
`barcode.pdf`	`$LOCALTEXMF/doc/pstricks/`

There is only one macro `\psbarcode` with the usual PSTricks syntax

```
\psbarcode[<TeX options>]{<PS options>}{<bar type>}
```

Important is the fact, that the barcode is printed in a box of zero dimension. If you want to save some space in your text, use the `pspicture` environment or the `\makebox` macro.

2 The options

2.1 The TEX options

name	*default*	*remarks*
`transx`	0	horizontal shift
`transy`	0	vertical shift
`scalex`	1	horizontal scaling
`scaley`	1	vertical scaling
`rotate`	0	rotating angle in degrees

2.2 The PostScript options

name	*default*	*remarks*
`height`	1	dimension is inch
`textsize`	10	dimension is pt
`textpos`	-2	dimension is pt; it is the shift for additional code text
`inkspread`	0.15	dimension is pt
`showborder`	-	-
`borderwidth`	0.5	dimension in pt
`borderleft`	10	dimension in pt
`borderright`	10	dimension in pt
`bordertop`	1	dimension in pt
`borderbottom`	1	dimension in pt
`borderwidth`	0.5	dimension in pt
`width`	-	dimension in inch
`font`	/Helvetica	must be a PostScript font
`includetext`	-	enable human readable text
`includecheck`	-	enable check digit
`includecheckintext`	-	check digit visible in text

2.3 Examples for the TeX options

```
\begin{pspicture}(3.5,1.2in)
\psbarcode{12345678}{includetext}{ean8}
\end{pspicture}
\begin{pspicture}(-2,-1.5)(0.5,0.2in)
\psbarcode[rotate=180,linecolor=red]{12345678}{includetext guardwhitespace height=0.6}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode[scalex=1.5,scaley=0.5,transy=20]{12345678}{includetext inkspread=0.5}{ean8}
\end{pspicture}
```

2.4 Examples for the PostScript options

```
\begin{pspicture}(3.5,1.2in)
\psbarcode{12345678}{includetext guardwhitespace height=0.6}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode{12345678}{textsize=15 includetext guardwhitespace height=0.6}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode{12345678}{includetext inkspread=0.5}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode{12345678}{includetext textpos=0}{ean8}
\end{pspicture}
```

```
\begin{pspicture}(3.5,1.2in)
\psbarcode{12345678}{includetext guardwhitespace}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode{12345678}{textsize=15 includetext guardwhitespace width=2}{ean8}
\end{pspicture}
```

3 Possible barcodes

The following section shows the symbologies that are supported by the encoders, including the available features for each. This list may not be up-to-date. If it does not contain any of the formats or features that you require then check the project source code or try the support mailing list.

3.1 EAN-13

Characters `0123456789`

Data 12 or 13 digits

Options

Option	Feature
`includetext`	Enable human readable text

Notes If just 12 digits are entered then the check digit is calculated automatically

9 781860 742712 >

```
\begin{pspicture}(3,1.2in)
\psbarcode[scalex=0.8,scaley=0.8]{9781860742712}{includetext
  guardwhitespace}{ean13}
\end{pspicture}
```

3.2 EAN-8

Characters `0123456789`

Data 8 digits

Options

Option	Feature
`includetext`	Enable human readable text

< 1234 5678 >

```
\begin{pspicture}(-2,-1.2)(0,0.2in)
\psbarcode[rotate=180,linecolor=red]{12345678}{includetext
  guardwhitespace height=0.6}{ean8}
\end{pspicture}
```

3.3 UPC-A

Characters `0123456789`

Data 11 or 12 digits

Options

Option	Feature
`includetext`	Enable human readable text

Notes If just 11 digits are entered then the check digit is calculated automatically

```
\begin{pspicture}(3,1.2in)
\psbarcode[transx=15pt,transy=10pt]{78858101497}{includetext}{upca}
\qdisk(0,0){3pt}\rput[lb](5pt,-10pt){Origin}
\end{pspicture}
```

● Origin

3.4 UPC-E

Characters `0123456789`

Data 7 or 8 digits

Options

Option	Feature
`includetext`	Enable human readable text

Notes If just 7 digits are entered then the check digit is calculated automatically

```
\begin{pspicture}(1.5,1.2in)
\psbarcode{0123456}{includetext}{upce}
\end{pspicture}
```

3.5 EAN-5

Characters `0123456789`

Data 5 digits

Options

Option	Feature
`includetext`	Enable human readable text

```
\begin{pspicture}(2,1in)
\psbarcode{90200}{includetext guardwhitespace}{ean5}
\end{pspicture}
```

3.6 EAN-2

Characters `0123456789`

Data 2 digits

Options

Option	Feature
`includetext`	Enable human readable text

38>

```
\begin{pspicture}(1,1in)
\psbarcode{38}{includetext guardwhitespace}{ean2}
\end{pspicture}
```

3.7 ISBN

Characters `-0123456789`

Data 9 or 10 digits seperated appropriately with dashes

Options

Option	Feature
`includetext`	Enable human readable text

Notes If just 9 digits are entered then the human readable ISBN check digit is calculated automatically

ISBN 1-58880-149-7

```
\begin{pspicture}(3,1in)
\psbarcode{1-58880-149}{includetext guardwhitespace}{isbn}
\end{pspicture}
```

9 781588 801494 >

3.8 Code-39

Characters `0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ-. $/`

Data Variable number of characters, digits and any of the symbols `-. *$/+%`.

Options

Option	Feature
`includecheck`	Enable check digit
`includetext`	Enable human readable text
`includecheckintext`	Make check digit visible in text

* C O D E - 3 9 *

```
\begin{pspicture}(5,1in)
\psbarcode{CODE-39}{includecheck includetext}{code39}
\end{pspicture}
```

3.9 Code-128 and UCC/EAN-128

Characters !"#$%&'\(\)*+,-./0...9:;<=>?@A...Z[\\]^_‘a...z{|}~

Data Variable number of ASCII characters and special funtion symbols, starting with the approriate start character for the initial character set. UCC/EAN-128s must have a manditory FNC 1 symbol immediately following the start character.

Options

Option	Feature
includetext	Enable human readable text
includecheckintext	Make check digit visible in text

Notes Any non-printable character can be entered via its escaped ordinal value, for example ^070 for ACK and ^102 for FNC 1. Since a caret symbol serves as an escape character it must be escaped as ^062 if used in the data. The check character is always added automatically.

C o u n t 1234 !

```
\begin{pspicture}(5,1in)
\psbarcode{^104^102Count^0991234^101!}{includetext}{
  code128}
\end{pspicture}
```

3.10 Rationalized Codabar

Characters 0123456789-$:/.+ABCD

Data Variable number of digits and any of the symbols -$:/.+ABCD.

Options

Option	Feature
includecheck	Enable check digit
includetext	Enable human readable text
includecheckintext	Make check digit visible in text

0 1 2 3 4 5 6 7 8 9

```
\begin{pspicture}(4,1in)
\psbarcode{0123456789}{includetext}{
  rationalizedCodabar}
\end{pspicture}
```

3.11 Interleaved 2 of 5 and ITF-14

Characters `0123456789`

Data Variable number of digits. An ITF-14 is 14 characters and does not have a check digit.

Options

Option	Feature
`includecheck`	Enable check digit
`includetext`	Enable human readable text
`includecheckintext`	Make check digit visible in text

Notes The data may be automatically prefixed with 0 to make the data, including optional check digit, of even length.

```
\begin{pspicture}(5,0.7in)
\psbarcode{05012345678900}{includecheck height=0.7}{
  interleaved2of5}
\end{pspicture}
```

3.12 Code 2 of 5

Characters `0123456789`

Data Variable number of digits

Options

Option	Feature
`includetext`	Enable human readable text

0 1 2 3 4 5 6 7 8 9

```
\begin{pspicture}(5,1.2in)
\psbarcode{0123456789}{includetext textpos=75
  textfont=Helvetica textsize=16}{code2of5}
\end{pspicture}
```

3.13 Postnet

Characters `0123456789`

Data Variable number digits

Options

Option	Feature
`includetext`	Enable human readable text
`includecheckintext`	Make the check digit visible in the text

Notes Check digit is always added automatically

0 1 2 3 4 5 6 7

```
\begin{pspicture}(7,0.3in)
\psbarcode{01234567}{includetext textpos=-10
  textfont=Arial textsize=10}{postnet}
\end{pspicture}
```

3.14 Royal Mail

Characters `ZUVWXY501234B6789AHCDEFGNIJKLMTOPQRS`

Data Variable number digits and capital letters

Options

Option	Feature
`includetext`	Enable human readable text
`includecheckintext`	Make the check digit visible in the text

Notes Check digit is always added automatically

L E 2 8 H S 9 Z

```
\begin{pspicture}(5,0.5in)
\psbarcode{LE28HS9Z}{includetext}{royalmail}
\end{pspicture}
```

3.15 Kix (Customer index) – Dutch Mail

Characters `ZUVWXY501234B6789AHCDEFGNIJKLMTOPQRS`

Data Variable number digits and capital letters

Options

Option	Feature
`includetext`	Enable human readable text

Notes Check digit is always added automatically

1 2 0 3 A A 1 2

```
\begin{pspicture}(5,0.5in)
\psbarcode{1203AA12}{includetext}{kix}
\end{pspicture}
```

3.16 Australian postal service

Characters `ZUVWXY501234B6789AHCDEFGNIJKLMTOPQRSabc...xyz`

Data Variable number digits and letters

Options

Option	Feature
`includetext`	Enable human readable text
`includecheckintext`	Make the check digit visible in the text

3 9 5 4 9 5 5 4

```
\begin{pspicture}(5,0.5in)
\psbarcode{1139549554}{includetext}{auspost}
\end{pspicture}
```

3.17 Symbol

The purpose of the symbol encoder is to store the definitions of miscellaneous barcode symbols such as the FIM symbols used by the US Postal Service on their reply mail.

```
\begin{pspicture}(1cm,.5in)
\psbarcode{fima}{includetext}{symbol}
\end{pspicture}
\begin{pspicture}(1cm,.5in)
\psbarcode{fimb}{includetext}{symbol}
\end{pspicture}\\
\begin{pspicture}(1cm,.5in)
\psbarcode{fimc}{includetext}{symbol}
\end{pspicture}
\begin{pspicture}(1cm,.5in)
\psbarcode{fimd}{includetext}{symbol}
\end{pspicture}
```

3.18 MSI

Characters `01234B6789`

Data Variable number digits

Options

Option	Feature
`includecheck`	Enable check digit
`includetext`	Enable human readable text
`includecheckintext`	Make check digit visible in the text

0 1 2 3 4 5 6 7 8 9

```
\begin{pspicture}(6,1in)
\psbarcode{0123456789}{includecheck includetext}{msi}
\end{pspicture}
```

3.19 Plessey

Characters `01234B6789ABCDEF`

Data Variable number of hexadecimal characters

Options

Option	Feature
`includetext`	Enable human readable text
`includecheckintext`	Make the check digits visible in the text

Notes Check digits are always added automatically.

0 1 2 3 4 5 6 7 8 9 A B C D E F

```
\begin{pspicture}(12,1in)
\psbarcode{0123456789ABCDEF}{includetext}{plessey}
\end{pspicture}
```

4 Code Commentary

This commentary assumes familiarity with the PostScript language[1].

The code is split cleanly into two types of procedure:

The encoders Each of these represents a barcode symbology[2], e.g. EAN-13 or Code-128. It takes a string containing the barcode data and a string containing a list of options that modify the output of the encoder. It generates a structured representation of the barcode and its text for the symbology, including the calculation of check digits where necessary.

The renderer This takes the output of an encoder and generates a visual representation of the barcode.

This means that all barcodes can be generated simply in a similar manner:

```
(78858101497) (includetext height=0.6) upca barcode
(0123456789) (includecheck) interleaved2of5 barcode
```

[1]The PostScript Language Tutorial and Cookbook (a.k.a. the Blue Book), which is freely available online, serves as both a useful tutorial and reference manual to the language.

[2]By symbology we mean an accepted standard for representation of data as a barcode

4.1 The Barcode Data Structure

The following table describes the structured representation of a barcode that is passed by an encoder to the renderer as a dictionary when the PostScript is executed.

Element	Key	Value
Space bar succession	`sbs`	String containing the integer widths, in points, of each bar and space, starting with the leftmost bar.
Bar height succession	`bhs`	Array containing the height of each bar in inches, starting with the leftmost bar.
Bar base succession	`bbs`	Array containing the offset of the base of each bar in inches, starting with the left-most bar.
Human readable text	`txt`	Array of arrays that contain the character, position, height, font and scale factor (font size), in points, for each of the visible text characters.

4.2 An Encoder

The procedure labelled code2of5 is a simple example of an encoder, which we will now consider. Its purpose is to accept as input a string containing the barcode contents and a string containing a list of options, and to process these in a way that is specific to this encoder, and finally to output an instance of the dictionary-based data structure described in section 4.1 that represents the barcode contents in the Code 2 of 5 symbology.

As with all of the encoders, the input string is assumed to be valid for the corresponding symbology, otherwise the behaviour is undefined.

The variables that we use in this procedure are confined to local scope by declaring the procedure as follows:

```
/code2of5 {

     0 begin

...

     end

} bind def
/code2of5 load 0 1 dict put
```

We start by immediately reading the contents strings that are passed as arguments to this procedure by the user.

```
     /options exch def
     /barcode exch def
```

We initialise a few default variables. Those variables corresponding to options that can be enabled with the options argument are initially set to false.

```
    /includetext false def
    /textfont /Courier def
    /textsize 10 def
    /textpos -7 def
    /height 1 def
```

The options string is tokenised with each successive token defining either a name value pair which we instantiate or a lone variable that we define as true, allowing us to override the given default variables given above.

```
options {
    token false eq {exit} if dup length string cvs (=) search
    true eq {cvlit exch pop exch def} {cvlit true def} ifelse
} loop
```

Since any user given options create variables that are strings we need to convert them back to their intended types.

```
/textfont textfont cvlit def
/textsize textsize cvr def
/textpos textpos cvr def
/height height cvr def
```

We then create an array of string encodings for each of the available characters which we then declare in another string. This information can be derived from careful reading of the relevant specification, although this is often surprisingly difficult to obtain.

```
    /encs
    [ (1111313111) (3111111131) (1131111131) (3131111111)
      (1111311131) (3111311111) (1131311111) (1111113131)
      (3111113111) (1131113111) (313111) (311131)
    ] def

    /barchars (0123456789) def
```

We now store the length of the content string and calculate the total number of bars and spaces in the resulting barcode. We initialise a string of size dependant on this length into which we will build the space bar succession. Similarly, we create an array into which we will add the human readable text information.

```
    /barlen barcode length def
    /sbs barlen 10 mul 12 add string def
    /txt barlen array def
```

We now begin to populate the space bar succession by adding the encoding of the start character to the beginning.

```
    sbs 0 encs 10 get putinterval
```

We now enter the main loop which iterates over the content string from start to finish, looking up the encoding for each character, adding this to the space bar succession.

It is important to understand how the encoding for a given character is derived. Firstly, given a character, we find its position in the string of all available characters. We then use this position to index the array of character encodings to obtain the encoding for the given character, which is added to the space/bar succession. Likewise, the character is added to the array of human readable text along with positioning and font information.

```
    0 1 barlen 1 sub {
        /i exch def
        barcode i 1 getinterval barchars exch search
        pop
        length /indx exch def
        pop pop
        /enc encs indx get def
        sbs i 10 mul 6 add enc putinterval
        txt i [barcode i 1 getinterval i 14 mul 10 add -7
               textfont textsize] put
    } for
```

The encoding for the end character is obtained and added to the end of the space bar succession.

```
    sbs barlen 10 mul 6 add encs 11 get putinterval
```

Finally we prepare to push a dictionary containing the space bar succession (and any additional information defined in section 4.1) that will be passed to the renderer.

```
     /retval 1 dict def
     retval (sbs) sbs put
     retval (bhs) [sbs length 1 add 2 idiv {height} repeat] put
     retval (bbs) [sbs length 1 add 2 idiv {0} repeat] put
     includetext {
         retval (txt) txt put
     } if
     retval
```

4.3 The Renderer

The procedure labelled barcode is known as the renderer, which we now consider. Its purpose is to accept as input an instance of the dictionary-based data structure described in section 4.1 that represents a barcode in some arbitrary symbology and produce a visual rendering of this at the current point.

The variables that we use in this procedure are confined to local scope by declaring the procedure as follows:

```
/barcode {

     0 begin

...

     end

} bind def
/barcode load 0 1 dict put
```

We then immediately read the dictionary-based data structure which is passed as a single argument to this procedure by an encoder, from which we extract the space bar succession, bar height succession and bar base succession.

```
    /args exch def
    /sbs args (sbs) get def
    /bhs args (bhs) get def
    /bbs args (bbs) get def
```

We attempt to extract from the dictionary the array containing the information about human readable text. However, this may not exist in the dictionary in which case we create a default empty array.

```
    args (txt) known
    {
        /txt args (txt) get def
    }
    {
        /txt [] def
    } ifelse
```

We have extracted or derived all of the necessary information from the input, and now use the space bar succession, bar height succession and bar base succession in calculations that create a single array containing elements that give coordinates for each of the bars in the barcode.

We start by creating a bars array that is half the length of the space bar succession. We build this by repeatedly adding array elements that contain the height, x-coordinate, y-coordinate and width of single bars. The height and y-coordinates are read from the bar height succession and the bar base succession, respectively, whilst the x-coordinate and the width are made from a calculation of the total indent, based on the space bar succession and a compensating factor that accounts for ink spread.

```
    /bars sbs length 1 add 2 idiv array def
    /x 0.00 def
    0 1 sbs length 1 sub {
        /i exch def
        /d sbs i get 48 sub def
        i 2 mod 0 eq
        {
            /h bhs i 2 idiv get 72 mul def
            /c d 2 div x add def
            /y bbs i 2 idiv get 72 mul def
            /w d 0.15 sub def
            bars i 2 idiv [h c y w] put
        } if
        /x x d add def
    } for
```

Finally, we perform the actual rendering in two phases. Firstly we use the contents of the bars array that we just built to render each of the bars, and secondly we use the contents of the text array extracted from the input argument to render the text. We make an efficiency saving here by not performing loading and rescaling of a font if the scale factor for the font size is 0. The graphics state is preserved across calls to this procedure to prevent unexpected interference with the users environment.

```
    gsave

    bars {
        {} forall
        setlinewidth moveto 0 exch rlineto stroke
    } forall

    txt {
        {} forall
        dup 0 ne {exch findfont exch scalefont setfont}
        {pop pop}
        ifelse
        moveto show
    } forall

    grestore
```

4.4 Notes Regarding Coding Style

PostScript programming veterans are encouraged to remember that the majority of people who read the code are likely to have little, if any, prior knowledge of the language.

To encourage development, the code has been written with these goals in mind:

- That it be easy to use and to comprehend
- That it be easy to modify and enhance

To this end the following points should be observed for all new code submissions:

- New encoders should be based on the code of a similar existing encoder
- Include comments where these clarify the operations involved, particular where something unexpected happens
- Prefer simplicity to efficency and clarity to obfuscation, except where this will be a problem

4.5 Installing the Barcode Generation Capability into a Printer's Virtual Machine

Most genuine PostScript printers allow procedures to be defined such that they persist across different jobs through the use of the `exitserver` command. If your printer supports this then you will be able to print the main code containing the definitons of all the encoders and the renderer once, e.g. soon after the device is turned on, and later omit these definitons from each of the barcode documents that you print.

To install the barcode generation capabilities into the virtual machine of a PostScript printer you need to uncomment a line near the top of the code so that it reads:

```
serverdict begin 0 exitserver
```

Once this code is printed the procedural definitions for the encoders and the renderer will remain defined across all jobs until the device is reset either by power-cycling or with the following code:

```
serverdict begin 0 exitserver systemdict /quit get exec
```

5 Package code

5.1 pst-barcode.sty

```
\RequirePackage{pstricks}
\ProvidesPackage{pst-barcode}[2005/07/24 package wrapper for
  pst-barcode.tex (hv)]
\input{pst-barcode.tex}
\ProvidesFile{pst-barcode.tex}
  [\filedate\space v\fileversion\space `PST-barcode' (hv)]
\endinput
```

5.2 pst-barcode.tex

```
%%
%% This is file `pst-barcode.tex',
%%
%% IMPORTANT NOTICE:
%%
%% Package `pst-barcode.tex'
%%
%% Terry Burton <tez _at_ terryburton.co.uk>
%% Herbert Voss <voss _at_ perce.de>
%%
%% This program can be redistributed and/or modified under the terms
%% of the LaTeX Project Public License Distributed from CTAN archives
%% in directory macros/latex/base/lppl.txt.
%%
%% DESCRIPTION:
%%   `pst-barcode' is a PSTricks package to draw barcodes, whatelse ...
%%
\csname PSTBarcodeLoaded\endcsname
\let\PSTBarcodeLoaded\endinput
\ifx\PSTricksLoaded\endinput\else\input pstricks.tex\fi
%
\def\fileversion{0.03}
\def\filedate{2005/09/17}
\message{`PST-barcode' v\fileversion, \filedate\space (tb,hv)}
%
\edef\PstAtCode{\the\catcode`\@} \catcode`\@=11\relax
\ifx\PSTXKeyLoaded\endinput\else\input pst-xkey \fi
\pst@addfams{pst-barcode}
\pstheader{pst-barcode.pro}
%
%
\define@key[psset]{pst-barcode}{transx}{%
  \pst@getlength{#1}\psk@barcode@transx}
\define@key[psset]{pst-barcode}{transy}{%
  \pst@getlength{#1}\psk@barcode@transy}
\define@key[psset]{pst-barcode}{scalex}{\def\psk@barcode@scalex{#1}}
\define@key[psset]{pst-barcode}{scaley}{\def\psk@barcode@scaley{#1}}
\define@key[psset]{pst-barcode}{rotate}{%
  \pst@getangle{#1}\psk@barcode@rotate}
\psset[pst-barcode]{transx=0,transy=0,scalex=1,scaley=1,rotate=0}
```

```
%
\def\psbarcode{\pst@object{psbarcode}}
\def\psbarcode@i#1#2#3{%
  \begin@SpecialObj
  \addto@pscode{
    gsave
    \psk@barcode@rotate\space
    \psk@barcode@scalex\space \psk@barcode@scaley\space
    \psk@barcode@transx\space \psk@barcode@transy\space
    translate scale rotate
    (#1) (#2) #3\space barcode
    grestore
  }%
  \end@SpecialObj%
  \ignorespaces%
}
%
\catcode`\@=\PstAtCode\relax
%
%% END: pst-barcode.tex
\endinput
```