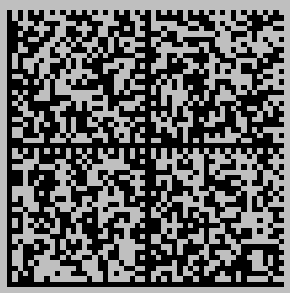
PSTricks

pst-barcode

A PSTricks package for drawing bar codes; v.0.11

October 7, 2013



Documentation by
Herbert Voß

Package author(s):
Terry Burton
Herbert Voß

Contents

1 Introduction

The pstricks package provides (essentially) one macro for printing barcodes. The type of the code is defined by a parameter and passed to postscript. To install the package put the three files in a place, where TEX will search for the files:

name	*meaning*	*target dir*
pst-barcode.tex	LATEX style file – wrapper	$LOCALTEXMF/tex/generic/pstricks/
pst-barcode.sty	TEX file – PS interface	$LOCALTEXMF/tex/latex/pstricks/
pst-barcode.pro	PostScript file	$LOCALTEXMF/dvips/pstricks/
pst-barcode-doc.tex	documentation source	$LOCALTEXMF/doc/pstricks/
pst-barcode-doc.bib	bibliography source	$LOCALTEXMF/doc/pstricks/
pst-barcode-doc.pdf	documentation	$LOCALTEXMF/doc/pstricks/

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

```
\psbarcode[Options]{PS options}{barcode type}
```

Important is the fact, that the barcode is printed in a TEX 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

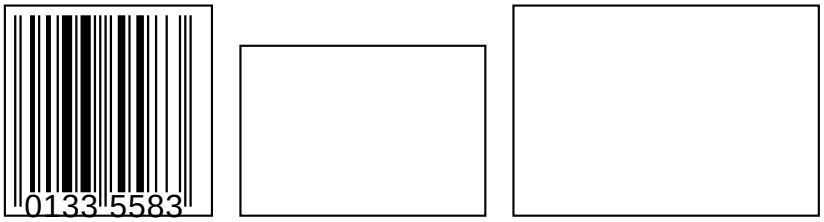
2.1 The TEX options

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
`parse`	–	parse variable field für decimal values, like ^032 for space, and convert them to ASCII

2.3 Examples for the TeX options




```
\psframebox{\begin{pspicture}(2.5,1in)
\psbarcode{01335583}{includetext}{ean8}
\end{pspicture}}\quad
\psframebox{\begin{pspicture}(-2.6,-1.5)(0.4,0.2in)
%\psbarcode[rotate=180,linecolor=red]{01335583}{includetext guardwhitespace height=0.6}{ean8}
\end{pspicture}}\quad
\psframebox{\begin{pspicture}(3.8,1in)
%\psbarcode[scalex=1.5,scaley=0.5,transy=1]{01335583}{includetext inkspread=0.5}{ean8}
\end{pspicture}}
```

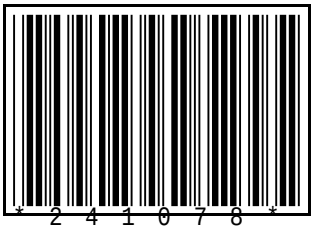
2.4 Examples for the PostScript options


```
\begin{pspicture}(3.5,1.2in)
\psbarcode{01335583}{includetext guardwhitespace height=0.6}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode{01335583}{textsize=15 includetext guardwhitespace height=0.6}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode{01335583}{includetext inkspread=0.5}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode{01335583}{includetext textpos=0}{ean8}
\end{pspicture}
```

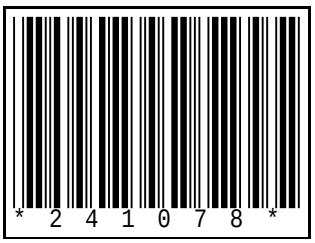

```
\begin{pspicture}(3.5,1.2in)
\psbarcode{01335583}{includetext guardwhitespace}{ean8}
\end{pspicture}
\begin{pspicture}(3.5,1.2in)
\psbarcode{01335583}{textsize=15 includetext guardwhitespace width=2}{ean8}
\end{pspicture}
```

3 Usage

By default the barcode has a width and a height of zero. Using the `\parbox` macro or the `pspicture` environment can reserve the needed space for the barcode. The `\fbox` in the following examples is used only for demonstration.

foo  bar

```
foo
\fbox{%
  \parbox[b][1in]{1.5in}{\psbarcode{241078}{includetext width=1.5 height=1}{code39}}}
bar
```

foo  bar

```
foo
\fbox{%
\begin{pspicture}(0,-8pt)(1.5in,1in)
  \psbarcode{241078}{includetext width
    =1.5 height=1}{code39}
\end{pspicture}}
bar
```

4 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.

4.1 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

```
\begin{pspicture}(3,1.2in)
\psbarcode[scalex=0.8,scaley=0.8]{9783865415561}{includetext guardwhitespace}{ean13}
\end{pspicture}
```

4.2 EAN-8

Characters `0123456789`

Data 8 digits

Options

Option	Feature
`includetext`	Enable human readable text


```
\begin{pspicture}(-2,-1.2)(0,0.2in)
\psbarcode[rotate=180,linecolor=red]{01335583}{includetext guardwhitespace height
  =0.6}{ean8}
\end{pspicture}
```

4.3 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}
```

4.4 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}
```

4.5 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}
```

4.6 EAN-2

Characters `0123456789`

Data 2 digits

Options

Option	Feature
`includetext`	Enable human readable text


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

4.7 ISBN

An ISBN symbol is really an EAN-13 with a particular prefix, 978 for the older ISBN-10 format, and others for the new ISBN-13 format.

Characters `-0123456789`

Data 9 or 10 digits for ISBN-10 seperated appropriately with dashes

Data 12 or 13 digits for ISBN-13 seperated appropriately with dashes

Options

Option	Feature
`includetext`	Enable human readable text

Notes If just 9 (ISBN-10) or 12 (ISBN-13) digits are entered then the human readable, ISBN check digit is calculated automatically

ISBN 978-3-86541-114-3


```
\begin{pspicture}(3,1in)
\psbarcode{3-86541-114}{includetext guardwhitespace}{isbn
  }
\end{pspicture}
```

ISBN 978-3-86541-114-3


```
\begin{pspicture}(3,1in)
\psbarcode{978-3-86541-114}{includetext guardwhitespace}{
  isbn}
\end{pspicture}
```

4.8 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


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

4.9 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.


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

4.10 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


```
\begin{pspicture}(4,1in)
\psbarcode{A0123456789B}{includetext}{
  rationalizedCodabar}
\end{pspicture}
```

4.11 Interleaved 2 of 5 and ITF-14

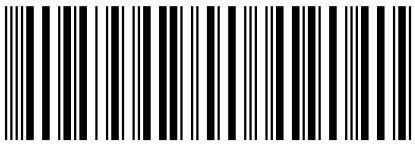
Characters 0123456789

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}
```

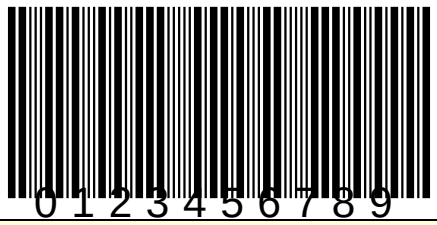
4.12 Code 2 of 5

Characters 0123456789

Data Variable number of digits

Options

Option	Feature
includetext	Enable human readable text




```
\begin{pspicture}(5,1.2in)
\psbarcode{0123456789}{includetext textpos=75 textfont=Helvetica textsize=16}{code2of
  5}
\end{pspicture}
```

4.13 Postnet

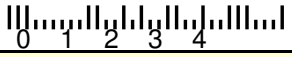
Characters 0123456789

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




```
\begin{pspicture}(7,0.3in)
\psbarcode{01234}{includetext textpos=-10 textfont=Arial textsize=10}{postnet}
\end{pspicture}
```

4.14 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


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

4.15 Kix (Customer index) – Dutch Mail

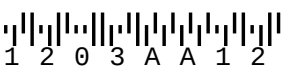
Characters ZUVWXY501234B6789AHCDEFGNIJKLMTOPQRS

Data Variable number digits and capital letters

Options

Option	Feature
includetext	Enable human readable text

Notes Check digit is always added automatically




```
\begin{pspicture}(5,0.3in)
\psbarcode{1203AA12}{includetext}{kix}
\end{pspicture}
```

4.16 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


```
\begin{pspicture}(5,0.3in)
\psbarcode{1139549554}{includetext}{auspost}
\end{pspicture}
```

4.17 Japan post service

Characters `ZUVWXY501234B6789AHCDEFGNIJKLMTOPQRSabc...xyz`

Data Variable number digits and letters

Options

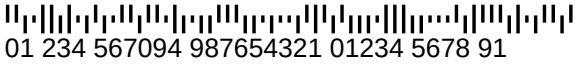
Option	Feature
`includetext`	Enable human readable text
`includecheckintext`	Make the check digit visible in the text


```
\begin{pspicture}(0,-5mm)(7,0.5in)
\psbarcode{6540123789-A-K-Z}{includetext textxalign=center}{japanpost}
\end{pspicture}
```

4.18 onecode

Characters `0123456789`

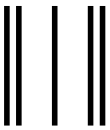
Data Variable number digits



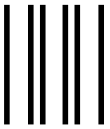

```
\begin{pspicture}(0,-5mm)(5,0.3in)
\psbarcode{01234567094987654321012345678 91}{includetext}{onecode}
\end{pspicture}
```

4.19 Symbol

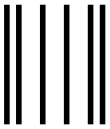
The purpose of the symbol encoder is to store the definitions of miscellaneous barcode symbols such as the FIM symbols used by the US Postal Service on their reply mail.



```
\begin{pspicture}(1cm,1.5cm)
\psbarcode{fima}{}{symbol}
\end{pspicture}
```



```
\begin{pspicture}(1cm,1.5cm)
\psbarcode{fimb}{}{symbol}
\end{pspicture}
```



```
\begin{pspicture}(1cm,1.5cm)
\psbarcode{fimc}{}{symbol}
\end{pspicture}
```

```
\begin{pspicture}(1cm,1.5cm)
\psbarcode{fimd}{}{symbol}
\end{pspicture}
```

4.20 MSI

Characters `0123456789`

Data Variable number digits

Options

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


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

4.21 Plessey

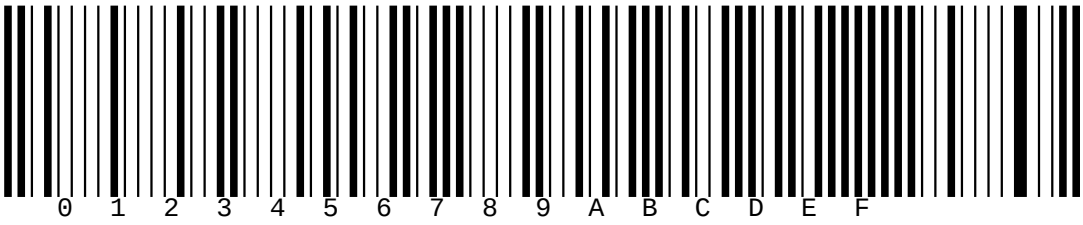
Characters `01234B6789ABCDEF`

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.




```
\begin{pspicture}(0,-2mm)(12,1in)
\psbarcode{0123456789ABCDEF}{includetext}{plessey}
\end{pspicture}
```

4.22 Reduced Space Symbology (RSS)

This is a family that includes RSS-14®, RSS Limited®, and RSS Expanded® (http://www.gs1.org/productssolutions/barcodes/technical/rss.html). RSS-14 and RSS Limited encode Global Trade Item Numbers (GTINs). RSS Expanded will encode any of the GS1 System identification numbers as well as all other Application Identifiers. RSS is used in the healthcare industry.

Characters 0123456789

Data Variable number digits

rss14 (databaromni)


```
\begin{pspicture}(12,.3in)
\psbarcode{(01)24012345678905}{format=truncated includetext height=0.3}{databaromni}
\end{pspicture}
```

rsslimited (databarlimited)

```
\begin{pspicture}(12,.3in)
\psbarcode{(01)15012345678907}{height=0.3}{databarlimited
  }
\end{pspicture}
```

rssexpanded (databarexpanded)

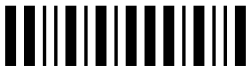
```
\begin{pspicture}(12,.3in)
\psbarcode{(10)12A}{height=0.3}{databarexpanded}
\end{pspicture}
```

4.23 Pharmacode

For a documentation see http://www.laetus.com/laetus.php?request=file&id=69.

Characters 0123456789

Data Variable number digits



```
\begin{pspicture}(12,.3in)
\psbarcode{117480}{}{pharmacode}
\end{pspicture}
```

4.24 PDF417

For a documentation see http://de.wikipedia.org/wiki/Strichcode#PDF417.

Characters 0123456789

Data Variable number characters

```
\begin{pspicture}(2in,0.3in)
\psbarcode{^453^178^121^239}{columns=2 rows=10}{pdf417}
\end{pspicture}
```

4.25 Data matrix

For a documentation see http://de.wikipedia.org/wiki/Strichcode#DataMatrix. It is a 2D matrix-style barcode that can encode full 256 character extended-ASCII. Also known as: Data Matrix ECC 200. Variants:

GS1 DataMatrix is a variant of Data Matrix that should be used when encoding data that is in GS1 Application Identifier standard format.

HIBC Data Matrix is a variant of Data Matrix that should be used when encoding HIBC formatted data.

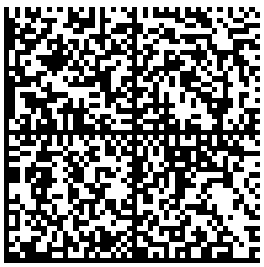
Standards: ISO/IEC 16022, ANSI/AIM BC11 ISS.
Data and Options

- The data field can contain any extended ASCII data.
- When the parse option is specified, any instances of `^NNN` in the data field are replaced with their equivalent ASCII value, useful for specifying unprintable characters.
- When the parsefnc option is specified, non-data function characters can be specified by `^FNC1`.
- The columns and rows options are used to specify the size of the symbol, either square or rectangular, one of: Square: 10x10, 12x12, 14x14, 16x16, 18x18, 20x20, 22x22, 24x24, 26x26, 32x32, 36x36, 40x40, 44x44, 48x48, 52x52, 64x64, 72x72, 80x80, 88x88, 96x96, 104x104, 120x120, 132x132, 144x144 Rectangular: 8x18, 8x32, 12x26, 12x36, 16x36, 16x48

- If the columns and rows are unspecified, the encoder will default to creating a (non-rectangular) symbol that is the minimum size to represent the given data.
- The raw option denotes that the data field is providing the input as a pre-encoded codewords in `^NNN` format, suitable for direct low-level encoding.
- The encoding option specifies how the data is to be encoded. Possible values are:
  - encoding=ascii - Extended ASCII data (default).
  - encoding=c40 - Optimized encoding for upper-case alphanumeric data. Can also encode extended ASCII data but incurs extra codeword overhead.
  - encoding=text - Optimized encoding for lower-case alphanumeric data. Can also encode extended ASCII data but incurs extra codeword overhead.
  - encoding=x12 - Optimized encoding restricted to upper-case alphanumeric data plus the characters `\r * >` and space.
  - encoding=raw - Same as the raw option.
- The prefix option allows adding a special codeword to the symbol prior to the data. Possible values are:
  - prefix=MAC5 - Prefixes the data with the 05 Macro codeword.
  - prefix=MAC6 - Prefixes the data with the 06 Macro codeword.
  - prefix=PROG - Prefixes the data with the reader programming codeword. May require encoding=c40, depending on the reader.
  - prefix=FNC1 - Prefixes the data with the FNC1 codeword.

Characters extended ASCII

Data Variable number characters



```
\begin{pspicture}(1.5in,1.5in)
\psbarcode{Herbert Voss ^142^164^186}{rows=48 columns=48 parse}{
  datamatrix}
\end{pspicture}
```

```
\begin{pspicture}(1in,1in)
\psbarcode{^098^099^100^142^164^186^101^102^103^104^105}{raw}{datamatrix
  }
\end{pspicture}
```

4.26 2D Maxi code

For a documentation see http://www.logicalconcepts.eu/wDeutsch/autoid/barcodetypen/index.html?navid=21. MaxiCode is a fixed-sized two-dimensional symbology created by the United Parcel Service that is primarily used for freight sortation and tracking. It's symbols have modules arranged in a hexagonal grid around a circular finder pattern which can be read omnidirectionally.

- MaxiCode has five alphabets A, B, C, D and E, each containing 64 characters.
  - Alphabet A contains mostly upper case letters, numbers and some common ASCII symbols.
  - Alphabet B contains mostly lower case letters and common ASCII symbols.
  - Alphabet C contains mostly upper case letters from the extended ASCII character set and less common ASCII symbols.
  - Alphabet D contains mostly lower case letters from the extended ASCII character set and less common ASCII symbols.
  - Alphabet E contains mostly the special ASCII characters and unprintable symbols.
- Non-printable/typable characters can be entered as their escaped ordinal values, e.g. ^028 for FS and ^059 for [shift B], etc.
- The symbol always starts in alphabet A which is suitable for the most basic contents.
- You can switch the working alphabet within the data using the [latch B], [latch A], ... characters.
- You can temporarily shift to another alphabet for a varying number of characters using the [shift A], [2 shift A], [3 shift A], ... characters.
- You can remain in an alphabet to which you have shifted using the [lock in C], [lock in D], ... characters.
- There are also more advanced features for encoding for which a thorough reading of the specification is required as these cannot be described succinctly here.

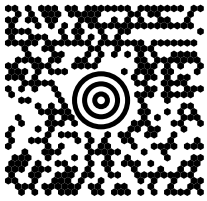
The modes:

- Mode 0 - Obsolete. (Older printers will produce Mode 0 if the firmware is outdated. Mode 0 MaxiCodes can be visually determined by examining the two horizontal hexagons in the upper right-hand corner. They will be white if the Mode is 0. For all other modes, they are black.)
- Mode 2 - Used for Numeric postal codes. (Primary use is US domestic destinations.)
- Mode 3 - Used for Alphanumeric postal codes. (Primary use is Int'l destinations.)

- Mode 4 - Standard Error Correction.
- Mode 5 - Enhanced Error Correction.
- Mode 6 - Used for programming hardware devices.

Characters `@ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789`

Data Variable number characters



```
\begin{pspicture}(1in,1in)
\expandafter\psbarcode{[\string\]>^03001^02996152382802^029840^029001^0291Z00004951^029UPSN^02906X
  610^029159^0291234567^0291/1^029^029Y^029634 ALPHA DR^029PITTSBURGH^029PA^029^004}{mode=2 parse
  }{maxicode}
\end{pspicture}
```

4.27 Aztec Code

Aztec Code is a 2 dimensional matrix style bar code symbology. Aztec Code was invented by Andrew Longacre (USA) in 1995 (http://de.wikipedia.org/wiki/Aztec_Code).

Characters `0123456789`

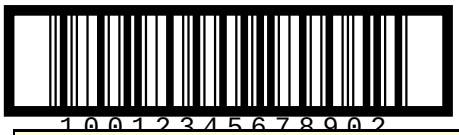
Data Variable number characters

```
\begin{pspicture}(1in,1in)
\psbarcode{0123456789}%
  {format=compact layers=3}{azteccode}
\end{pspicture}
```

4.28 itf14

Characters `0123456789`

Data Variable number characters




```
\begin{pspicture}(1in,1in)
\psbarcode{1001234567890}{showborder borderwidth=4 borderleft=15 borderright=15 height=0.5
  includecheck includetext includecheckintext textyoffset=-10}{interleaved2of5}
\end{pspicture}
```

4.29 QR Code

A QR Code (Quick Response) is a 2 dimensional matrix code created by Japanese corporation Denso-Wave in 1994 (http://de.wikipedia.org/wiki/QR_Code).

Characters `0123456789`

Data Variable number characters

The data field can contain any extended ASCII data. When the parse option is specified, any instances of `^NNN` in the data field are replaced with their equivalent ASCII value, useful for specifying unprintable characters.

The eclevel option is used to specify the error correction level:

eclevel=L - Low (default for micro format symbols)

eclevel=M - Medium (default for full format symbols)

eclevel=Q - Quality

eclevel=H - High

The eclevel will be opportunistically raised when this does not result in an increased symbol size. The encoding option is used to specify a desired encoding for the input data which can sometimes result in a more optimal symbol size:

encoding=alphanumeric - Alphanumeric data

encoding=numeric - Numeric data

encoding=byte - Byte based encoding

encoding=kanji - Kanji characters based on Shift JIS

encoding=raw - Equivalent to the raw option If left unspecified the optimal available encoding will be chosen for the given data in the following order of preference: numeric, alphanumeric, kanji, byte. The version option is used to specify the size of the symbol, 1 to 40 for full format symbols or `version=M1`, `version=M2`, `version=M3` or `version=M4` for micro format symbols. If unspecified, the encoder will select the version of the symbol that is the minimum size to represent the given data at the selected error correction level. The format option is used to select between format=full and format=micro symbol types. By default, full format symbols will be generated. The raw option denotes that the data field is providing the input as a pre-encoded bitstream (excluding the terminator bits) suitable for direct low-level encoding.

```
\begin{pspicture}(1in,1in)
\psbarcode{http://www.dante.de}{}{qrcode}
\end{pspicture}
```

```
\begin{pspicture}(1in,1in)
\psbarcode{http://www.dante.de}{eclevel=M}{qrcode}
\end{pspicture}
```

```
\begin{pspicture}(1in,1in)
\psbarcode{QR ^067ode}{parse}{qrcode}
\end{pspicture}
```

```
\begin{pspicture}(2in,2in)
\psbarcode{QR CODE 1234}{version=10 eclevel=Q}{qrcode}
\end{pspicture}
```

```
\begin{pspicture}(0.5in,0.5in)
\psbarcode{01234567}{format=micro}{qrcode}
\end{pspicture}
```

```
\begin{pspicture}(1in,1in)
\psbarcode{00010000001000000000110001010110011000011000}{raw}{qrcode}
\end{pspicture}
```

```
\begin{pspicture}(0.5in,0.5in)
\psbarcode{00100000000001100010101100110101001101110000101001110101001 01}{format=micro raw}{qrcode
  }
\end{pspicture}
```

5 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
```

5.1 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.

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

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 leftmost 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.

5.2 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 5.1 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 5.1) 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
```

5.3 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 5.1 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
```

5.4 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

5.5 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
```

6 List of all optional arguments for pst-barcode

Key	Type	Default
transx	ordinary	0
transy	ordinary	0
scalex	ordinary	1
scaley	ordinary	1
rotate	ordinary	0

References

[1] Hendri Adriaens. xkeyval package. CTAN:/macros/latex/contrib/xkeyval, 2004.

[2] Denis Girou. Présentation de PSTricks. *Cahier GUTenberg*, 16:21–70, April 1994.

[3] Michel Goosens, Frank Mittelbach, Sebastian Rahtz, Denis Roegel, and Herbert Voß. *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Mass., 2007.

[4] Alan Hoenig. *TeX Unbound: LaTeX & TeX Strategies, Fonts, Graphics, and More*. Oxford University Press, London, 1998.

[5] Laura E. Jackson and Herbert Voß. Die plot-funktionen von pst-plot. *Die TeXnische Komödie*, 2/02:27–34, June 2002.

[6] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. IWT, Vaterstetten, 1989.

[7] Frank Mittelbach and Michel Goosens et al. *The LaTeX Companion*. Addison-Wesley Publishing Company, Boston, second edition, 2004.

[8] Frank Mittelbach and Michel Goosens et al. *Der LaTeX Begleiter*. Pearson Education, München, zweite edition, 2005.

[9] Herbert Voß. *Chaos und Fraktale selbst programmieren: von Mandelbrotmengen über Farbmanipulationen zur perfekten Darstellung*. Franzis Verlag, Poing, 1994.

[10] Herbert Voß. Die mathematischen Funktionen von PostScript. *Die TeXnische Komödie*, 1/02, March 2002.

[11] Herbert Voß. *PSTricks Grafik für TeX und LaTeX*. DANTE – lehmanns media, Heidelberg/Hamburg, 6. edition, 2010.

[12] Herbert Voß. *LaTeX Referenz*. DANTE – lehmanns media, Heidelberg/Hamburg, 2. edition, 2010.

[13] Herbert Voß. *LaTeX Quick Reference*. UIT, Cambridge/UK, 1. edition, 2011.

[14] Herbert Voß. *PSTricks – Graphics for LaTeX*. UIT, Cambridge/UK, 1. edition, 2011.

[15] Herbert Voß. *Presentations with LaTeX*. DANTE – Lehmanns Media, Heidelberg/Berlin, 1. edition, 2012.

[16] Timothy Van Zandt. *PSTricks - PostScript macros for generic TeX*. http://www.tug.org/application/PSTricks, 1993.

[17] Timothy Van Zandt. *multido.tex - a loop macro, that supports fixed-point addition*. CTAN:/graphics/pstricks/generic/multido.tex, 1997.

[18] Timothy Van Zandt. *pst-plot: Plotting two dimensional functions and data*. CTAN:graphics/pstricks/generic/pst-plot.tex, 1999.

[19] Timothy Van Zandt and Denis Girou. Inside PSTricks. *TUGboat*, 15:239–246, September 1994.

Index