NAME

barcode − a stand alone program to run the barcode library

SYNOPSIS

barcode [−b - | string] [−e encoding] [−o - | outfile] [−w N] [−h N] [−x N] [−y N] [−l N] [−c N]

DESCRIPTION

The information below is extracted from the texinfo file, which is the preferred source of information.

The **barcode** program is a front-end to access some features of the library from the command line. It is able to read user supplied strings from the command line or a data file (standard input by default) and encode all of them.

OPTIONS

barcode accepts the following options:

--help or -h
: Print a usage summary and exit.

-i filename
: Identify a file where strings to be encoded are read from. If missing (and if -b is not used) it defaults to standard input. Each data line of the input file will be used to create one barcode output.

-o filename
: Output file. It defaults to standard output.

-b string
: Specify a single ‘‘barcode’’ string to be encoded. The option can be used multiple times in order to encode multiple strings (this will result in multi-page postscript output or a table of barcodes if -t is specified). The strings must match the encoding chosen; if it doesn’t match the program will print a warning to stderr and generate ‘‘blank’’ output (although not zero-length). Please note that a string including spaces or other special characters must be properly quoted.

-e encoding
: **encoding** is the name of the chosen encoding format being used. It defaults to the value of the environment variable BARCODE_ENCODING or to auto detection if the environment is also unset.

-g geometry

The geometry argument is of the form ‘‘[<width> x <height>] [+ <xmargin> + <ymargin>]’’ (with no intervening spaces). Unspecified margin values will result in no margin; unspecified size results in default size. The specified values represent print points by default, and can be inches, millimeters or other units according to the -u option or the BARCODE_UNIT environment variable. The argument is used to place the printout code on the page. Note that an additional white margin of 10 points is added to the printout. If the option is unspecified, BARCODE_GEOMETRY is looked up in the environment, if missing a default size and no margin (but the default 10 points) are used.

-t table-geometry
: Used to print several barcodes to a single page, this option is meant to be used to print stickers. The argument is of the form ‘‘<columns> x <lines> [+ <leftmargin> + <bottommargin> [- <rightmargin> [- <topmargin>]]]’’ (with no intervening spaces); if missing, the top and right margin will default to be the same as the bottom and left margin. The margins are specified in print points or in the chosen unit (see -u below). If the option is not specified, BARCODE_TABLE is looked up in the environment, otherwise no table is printed and each barcode will get its own page.

-m margin(s)
: Specifies an internal margin for each sticker in the table. The argument is of the form ‘‘<xmargin>,<ymargin>’’ and the margin is applied symmetrically to the sticker. If unspecified, the environment variable BARCODE_MARGIN is used or a default internal margin of 10 points is used.

-n
: ‘‘Numeric’’ output: don’t print the ASCII form of the code, only the bars.

-c
: No checksum character (for encodings that allow it, like code 39, other codes, like UPC or EAN, ignore this option).

-E Encapsulated postscript (default is normal postscript. When the output is generated a EPS only one text string is encoded.

-p pagesize

Specify a non-default page size. The page size can be specified in millimeters, inches or plain numbers (for example: "210x297mm", "8.5x11in", "595x842"). A page specification as numbers will be interpreted according to the current unit specification (see -u below). If libpaper is available, you can also specify the page size with its name, like "A3" or "letter" (libpaper is a standard component of Debian GNU/Linux, but may be missing elsewhere). The default page size is your system-wide default if libpaper is there, A4 otherwise.

-u unit

Choose the unit used in size specifications. Accepted values are ‘‘mm’’, ‘‘cm’’, ‘‘in’’ and ‘‘pt’’. By default, the program will check BARCODE_UNIT in the environment, and assume points otherwise (this behaviour is compatible with 0.92 and previous versions. If -u appears more than once, each instance will modified the behaviour for the arguments at its right, as the command line is processes left to right. The program internally works with points, and any size is approximated to the nearest multiple of one point. The -u option affect -g (geometry), -t (table) and -p (page size).

ENCODING TYPES

The program encodes text strings passed either on the command line (with -b) or retrieved from standard input. The text representation is interpreted according to the following rules. When auto-detection of the encoding is enabled (i.e, no explicit encoding type is specified), the encoding types are scanned to find one that can digest the text string. The following list of supported types is sorted in the same order the library uses when auto-detecting a suitable encoding for a string.

UPC

The UPC frontend accepts only strings made up of digits (and, if a supplemental encoding is used, a blank to separate it). It accepts strings of 11 digits (UPC-A) or 6 digits (UPC-E). The 12th digit of UPC-A is the checksum and is added by the library, if you pass a 12-digit string it will be rejected as invalid. For UPC-A, a trailing string of 2 digits or 5 digits is accepted as well. Therefore, valid strings look like one of the following: ‘‘01234567890’’ (UPC-A), ‘‘012345’’ (UPC-E), ‘‘01234567890 12’’ (UPC-A, add-2) and ‘‘01234567890 12345’’ (UPC-A add-5).

EAN

The EAN frontend is similar to UPC; it accepts strings of digits, 12 or 7 characters long, the checksum digit is added by the library and a string of 13 or 8 characters is rejected. The add-2 and add-5 extension are accepted for the EAN13 encoding. Valid strings look like one of the following: ‘‘123456789012’’ (EAN-13), ‘‘1234567’’ (EAN-8), ‘‘123456789012 12’’ (EAN-13 with add-2) and ‘‘123456789012 12345’’ (EAN-13 with add-5).

ISBN

ISBN numbers are encoded as EAN-13 symbols, with an optional add-5 trailer. The ISBN frontend of the library accepts real ISBN numbers and deals with any hyphen and, if present, the ISBN checksum character before encoding data. Valid representations for ISBN strings are for example: ‘‘1-56592-292-1’’, ‘‘3-89721-122-X’’ and ‘‘3-89721-122-X 06900’’.

code 128-B

This encoding can represent all of the printing ASCII characters, from the space (32) to DEL (127). The checksum digit is mandatory in this encoding.

code 128-C

The ‘‘C’’ variation of Code-128 uses Code-128 symbols to represent two digits at a time (Code-128 is made

up of 104 symbols whose interpretation is controlled by the start symbol being used). Code 128-C is thus the most compact way to represent any even number of digits. The encoder refuses to deal with an odd number of digits because the caller is expected to provide proper padding to an even number of digits. (Since Code-128 includes control symbols to switch charset, it is theoretically possible to represent the odd digit as a Code 128-A or 128-B symbol, but this tool doesn't currently implement this option).

code 128 raw

Code-128 output represented symbol-by-symbol in the input string. To override part of the problems outlined below in specifying code128 symbols, this pseudo-encoding allows the used to specify a list of code128 symbols separated by spaces. Each symbol is represented by a number in the range 0-105. The list should include the leading character.The checksum and the stop character are automatically added by the library. Most likely this pseudo-encoding will be used with BARCODE_NO_ASCII and some external program to supply the printed text.

code 39

The code-39 standard can encode uppercase letters, digits, the blank space, plus, minus, dot, star, dollar, slash, percent. Any string that is only composed of such characters is accepted by the code-39 encoder. To avoid loosing information, the encoder refuses to encode mixed-case strings (a lowercase string is nonetheless accepted as a shortcut, but is encoded as uppercase).

interleaved 2 of 5

This encoding can only represent an even number of digits (odd digits are represented by bars, and even digits by the interleaving spaces). The name stresses the fact that two of the five items (bars or spaces) allocated to each symbol are wide, while the rest are narrow. The checksum digit is optional (can be disabled via BARCODE_NO_CHECKSUM). Since the number of digits, including the checksum, must be even, a leading zero is inserted in the string being encoded if needed (this is specifically stated in the specs I have access to).

code 128

Automatic selection between alphabet A, B and C of the Code-128 standard. This encoding can represent all ASCII symbols, from 0 (NUL) to 127 (DEL), as well as four special symbols, named F1, F2, F3, F4. The set of symbols available in this encoding is not easily represented as input to the barcode library, so the following convention is used. In the input string, which is a C-language null-terminated string, the NUL char is represented by the value 128 (0x80, 0200) and the F1-F4 characters are represented by the values 193-196 (0xc1-0xc4, 0301-0304). The values have been chosen to ease their representation as escape sequences.

Since the shell doesn't seem to interpret escape sequences on the command line, the "-b" option cannot be easily used to designate the strings to be encoded. As a workaround you can resort to the command echo, either within backticks or used separately to create a file that is then fed to the standard-input of barcode -- assuming your echo command processes escape sequences. The newline character is especially though to encode (but not impossible unless you use a csh variant.

These problems only apply to the command-line tool; the use of library functions doesn't give any problem. In needed, you can use the ‘‘code 128 raw’’ pseudo-encoding to represent code128 symbols by their numerical value. This encoding is used late in the auto-selection mechanism because (almost) any input string can be represented using code128.

Codabar

Codabar can encode the ten digits and a few special symbols (minus, plus, dollar, colon, bar, dot). The

characters ‘‘A’’, ‘‘B’’, ‘‘C’’ and ‘‘D’’ are used to represent four different start/stop characters. The input string to the barcode library can include the start and stop characters or not include them (in which case ‘‘A’’ is used as start and ‘‘B’’ as stop). Start and stop characters in the input string can be either all lowercase or all uppercase and are always printed as uppercase.

Plessey

Plessey barcodes can encode all the hexadecimal digits. Alphabetic digits in the input string must either be all lowercase or all uppercase. The output text is always uppercase.

MSI

MSI can only encode the decimal digits. While the standard specifies either one or two check digits, the current implementation in this library only generates one check digit.

BUGS

The current management of borders/margins is far from optimal. The ‘‘default’’ margin applied by the library interferes with the external representation, but I feel it is mandatory to avoid creating barcode output with no surrounding white space (the problem is especially relevant for EPS output).

EAN-128 is not (yet) supported. I plan to implement it pretty soon and then bless the package as version 1.0.

SEE ALSO

barcode(3)

AUTHORS

Alessandro Rubini <rubini@gnu.org> (maintainer)

Leonid A. Broukhis <leob@mailcom.com> (several encodings)